НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри
_____ Сергій СТІРЕНКО

“___”_____2020 р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Комп’ютерні системи та мережі»

спеціальності 123 «Комп’ютерна інженерія»

на тему: «Мобільний додаток - гра під IOS платформу»

Виконав:
студент IV курсу, групи ІО-64
Данило ПОЛІЩУК _____

Керівник:
Доцент, к.т.н.,
Віктор ПОРЄВ _____

Консультант з нормоконтролю:
Професор, д.т.н.,
Валерій СІМОНЕНКО _____

Рецензент:

_____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврьский)

Спеціальність – 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___» _____2020 р.

ЗАВДАННЯ
на дипломний проект студента

Поліщука Данила Олександровича

1. Тема проекту *«Мобільний додаток - гра під IOS платформу»*
керівник проекту Порєв Віктор Миколайович, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від «07» травня 2020 року №1081-с
2. Термін здачі студентом закінченого проекту
_____
3. Вихідні дані до проекту *технічна документація*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис та аналіз предметної області, аналіз аналогів у даній області, дослідження основних вимог до ПЗ, конструювання архітектури системи та створення компонентів системи .
5. Перелік графічного матеріалу
Принципова схема, функціональна схема та структурна схема

6. Консультанти проекту, з вказівкою розділів роботи, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
нормоконтроль	д.т.н., проф. Сімоненко В.П.		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів дипломного проекту	Строк виконання етапів проекту	Примітки
1.	*Затвердження теми роботи*	*1.09.2019*	
2.	*Вивчення та аналіз завдання*	*2.09.2019-11.03.2020*	
3.	*Розробка архітектури та загальної структури програми*	*11.03.2020-22.03.2020*	
4.	*Розробка структур окремих Інтерфейсів програми*	*22.03.2020-2.04.2020*	
5.	*Програмна реалізація*	*2.04.2020-15.04.2020*	
6.	*Оформлення пояснювальної записки*	*15.04.2020-21.05.2020*	
7.	*Захист програмного продукту*	*21.05.2020 – 25.05.2020*	
8.	*Передзахист*	*26.05.2020*	
9.	*Захист*		

Студент Данило ПОЛІЩУК

Керівник Віктор ПОРЄВ

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1.	А4		Завдання на дипломний проект	2	
2.	А4	ІАЛЦ.466500.002 ТЗ	Технічне завдання	4	
3.	А4	ІАЛЦ.466500.003 ПЗ	Пояснювальна записка	63	
4.	А4	ІАЛЦ.466500.004 А1	Принципова схема алгоритму	1	
5.	А4	ІАЛЦ.466500.005 А2	Функціональна схема	1	
6.	А4	ІАЛЦ.466500.006 А3	Структурна схема	1	

					ІАЛЦ.466500.001 ВП			
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Поліщук Д.О.			Мобільний додаток - гра під IOS платформу Відомість дипломного проекту	Літ.	Аркуш	Аркушів
Перевірив		Порєв В.М.					1	1
Реценз.						НТУУ «КПІ», ФІОТ, ІО-64		
Н. Контр.		Сімоненко В.П.						
Затв.								

Технічне завдання до дипломного проекту

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Технічне завдання спрямоване на розробку мобільного застосунку «Мобільний додаток – гра під iOS платформу». Область застосування: додаток для щоденного коротання часу через гру.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Завдання на виконання дипломного проекту освітньо-кваліфікаційного рівня «бакалавр», затвердженою кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного мобільного додатку є принесення задоволення користувачеві Призначення додатку: гра.

4. ДЖЕРЕЛА РОЗРОБКИ

На даний час існують багато рішень, як нативних так і мультиплатформних. Такі як Unity, Unreal Engine та інші, але ні один з сервісів не може запронувати такої легкості у використанні та такого тісного зв’зку з будь-яким з потенційно потрібних сервісів Apple.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розроблюваного продукту

- Захищеність даних.
- Можливість роботи програми разом з додатками з категорії «Музика»
- Можливість оплати внутрішньо ігрових покупок прямо в застосунку.

5.2. Вимоги до програмного забезпечення

- Операційна система iOS 10.0 і вище

5.3 Вимоги до апаратного забезпеення

- Мінімальні технічні вимоги:
  - Частота процессора: 1.3 ГГц.

- Мінімальний об’єм оперативної пам’яті: 1ГБ.
- Вільне місце не менш ніж 30МБ.

- Бажані технічні вимоги:
  - Частота процессора: 1.8 ГГц.
  - Об’єм оперативної пам’яті: 1.5ГБ.
  - Вільне місце не менш ніж 30МБ.

АНОТАЦІЯ

При виконанні даної дипломної роботи було розроблено програмний продукт для операційної системи iOS у якому розглядається система ігровий рушій SpriteKit та реалізація мобільного додатку-гри з його допомогою. Програмний продукт забезпечує користувача таким функціоналом, як: можливість грати, можливість переглядати набутий прогрес та статистику, можливість завантажувати шпалери у високій якості, можливість змінювати налаштувань, можливість переглядати та обирати рівні та героїв, можливість здобувати внутрішньо ігрові покупки.

Програмний продукт був створений на мові Swift у середовищі Xcode.

ABSTRACT

In the course of this thesis, a software product was developed for the iOS operating system, which examines the SpriteKit game propulsion system and implementation of a mobile game application with its help. The software product provides the user with such functionality as: the ability to play, the ability to view acquired progress and statistics, the ability to download wallpapers in high quality, the ability to change settings, the ability to view and select levels and characters, the ability to make in-app purchases.

The software was created in Swift in Xcode.

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту

на тему: «Мобільний додаток - гра під IOS платформу»

Київ – 2020

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ОС - Операційна Система

ПК - Персональний комп’ютер

UI - User Interface (користувальницький інтерфейс)

ВСТУП

Щороку люди починають копистуватися технікою все більше і більше, щороку почати користуватися технікою стає все легше і легше. Розвиваються технології, що спрощують користування людям з вадами слуху, зору, та інш. Ціна на потужний пристрій сьогодні доступніша ніж коли-завгодно за історію технологій. Глобальна диджиталізація - тільки початок розвитку нових та впевнених ріст вже існуючих ринків. А кількість користувачів тільки зростає. Гарним прикладом є різкий ріст користувачів мережі інтернет у Індії після реформ які значно зменшили ціни на інтернет.

Як наслідок - найбільш доступною всім у світі платформою є мобільні пристрої. Вони покривають все більше аспектів та єтапів нашого щоденного життя, що дозволяє єкономити час на рутинних або бюрократично-тривалих буденних справах накшталт оплати комунальних платежів, відвідування банків та інш.

Однією з галузей що з розвитком комп'ютерних технологій почали переносити у світ мобільних технологій є галузь ігор. В наш час неможливо уявити поїздку громадським транспортом без людей які весь час проводять у телефоні. А як краще можна скоротити умовні пів години у метро ніж за захоплюючою грою?

Завданням роботи є розробка мобільноо додатку-гри під платформу iOS. Результатом роботи є мобільний додаток-гра для платформи iOS.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Ринок мобільних ігор

На кінець 2019 року світовий ринок ігор оцінюється у 152 мільярди доларів, причому 45% від цього, 68,5 мільярда доларів, надходять безпосередньо з мобільних ігор. З цим приголомшливим зростанням (10,2% дпр, якщо бути точним) прийшов шум інвестицій та придбань, кожен бажаючий розрізати пиріг. Насправді, за останні 18 місяців у світовій ігровій індустрії спостерігалось інвестицій у розмірі 9,6 мільярдів доларів, і якщо інвестиції продовжуватимуться нинішніми темпами, обсяг інвестицій, сформованих у 2018-19-19 роках, буде більшим, ніж вісім попередніх років разом.

Сьогодні на мобільні ігри припадає 33% усіх завантажень програм, 74% споживчих витрат і 10% всього часу, проведеного через додаток. Прогнозується, що в 2019 році 2,4 мільярда людей будуть грати в мобільні ігри по всьому світу - це майже третина світового населення. Фактично, 50% користувачів мобільних додатків грають в ігри, що робить цю категорію додатків такою ж популярною, як музичні додатки, такі як Spotify та Apple Music, і поступається лише соціальним мережам та програмам комунікацій за витраченим часом.

В США час, витрачений на мобільні пристрої, також офіційно перевершив час телебачення - користувачі витрачають ще вісім хвилин на день на своїх мобільних пристроях. До 2021 року прогнозується, що ця кількість зросте до більш ніж 30 хвилин. Додатки - це новий простий час, а ігри захопили левову частку.

Доступність є найвищою, ніж вона коли-небудь була, оскільки перешкод для входу практично не існує. Від повсякденних ігор до недавнього виникнення надзвичайно популярного гіпер-випадкового жанру ігор, які швидко завантажуються, легко граються та піддаються тому, що їх можна грати на коротких сесіях протягом дня, грає майже кожен демографічний прошарок суспільства. Сьогодні середній вік мобільного геймера становить 36,3 (порівняно з 27,7 у 2014 році), гендерний розкол становить 51% жінок, 49% чоловіків, а третина всіх гравців у віці 36-50 років - далекий пласт від традиційного стереотипу «геймера».

Завдяки цим демографічним, географічним та споживчим змінам у мобільних екосистемах та ландшафтах розваг, не дивно, що ігровий простір привертає все більше уваги та інвестицій не лише з боку галузі, але останнім часом з традиційних фінансових ринків і навіть урядів.

Мобільний ринок іде семимильними кроками. І має потенціал значно збільшитися у 2020 році зявдяки новим тенденціям.

1. З розвитком 5G пришвидвиться розвиток хмарних ігор

Було б поспішно називати 2019 рік роком 5G, але все ж 2019 рік став роком початку глобального впровадження даного стандарту передачі даних.

2020 цілком ще може, не дивлячись на ситуацію з пандемією COVID 19, стати роком 5G. Адже впровадження технології 5G дійсно стає панівною тенденцією в області хмарної підтримки мобільних ігор.

2. Мобільні ігри з підтримкою хмарних сервісів дадуть більше можливого ігрового досвіду

У 2019 Google і Microsoft випустили свої амбітні хмарні ігрові рішення.

Мелісса Зелоф, віце-президент з маркетингу в IronSource, описує це:

«В цілому малоймовірно, що хмарні гри замінять консолі і ПК або будь-яким чином перетворять аудиторію казуальних гравців в фанатів хмарних рішень.

З мого досвіду, нові формати і технології на ринку мобільних ігор просто створять ще один прошарок, але не перевернуть весь ринок кардинально. Розширюючи діапазон можливого ігрового досвіду і створюючи більший доступ до традиційних ігор, хмарні рішення, ймовірно, але не обов'язково внесуть значний внесок в розвиток цього ринку».

3. Хардкорні геймери будуткупувани щомісячну підписку, а більш казуальні геймери- ні

У березні 2019 року Apple і Google оголосили про свої нові сервіси, які працюють за підпискою. Apple і Google пропонують користувачам підписку на ігрові сервіси - змінюючи дохід від ігрових покупок і реклами в додатках на щомісячну плату. Проте, не дивлячись на сформовану тенденцію переходу на нову систему монетизації ми все ще маємо безліч невідомих для повноцінних прогнозів на майбутнє: ні Apple, ні

Google, ні розробники, що працюють з будь-якої зі сторін, не пояснили повноцінну бізнес-модель даного напрямку.

Що ми дійсно знаємо, так це те, що підписні сервіси націлені більшою мірою на користувачів, які системно і постійно грають в ігри. Ті, хто приділяє іграм час тільки зрідка не є цільовою аудиторією нової стратегії монетизації ігор.

4. Гіпер-казуальщина ще міцніше закріпить за собою місце на ринку ігор

2019 рік був роком вибухового зростання для гіпер-казуальних ігор - IPM виріс на 70%. Безсумнівно, жанр гіпер-казуальних ігор досяг свого піку, і в 2020 році він, безумовно, стабілізується, оскільки користувачі більше не схильні до таких агресивних рекламних кампаній, як в минулому році. Але це не означає, що гіпер-казуальщина помре. Насправді, мені здається, що 2020 рік стане роком, в якому гіпер-казуальні ігри будуть позиціонуватися як повноцінний окремий жанр.

5. Паблішери об'єднають свої маркетингові напрямки

Сьогодні реклама все більше стає важливим джерелом доходу для розробників ігор. Фактично, App Annie прогнозує, що в 2020 році дохід від монетизації через показ реклами, нарешті, перевищить дохід від IAP (ігрові покупки всередині).

У 2019 ряд ігрових студій - в основному серед казуальних напрямків, почали об'єднувати свої UA відділи (User Acquisition - залучення та атрибуція платного мобільного трафіку) і монетизацію через ігрові покупки в єдину команду.

Метою цього об'єднання є забезпечення того, щоб одна людина або група фахівців працювали в тісному контакті, щоб побачити повну картину розвитку гри, що призведе до більш глибокого розуміння процесу монетизації кожного окремого користувача: каналу, який привів їх, і того, як вони взаємодіють з IAP і показами реклами в грі.

Отже ринок мобільних ігор як галузь є надзвичайно перспективною та упускати з виду її точно не варто, а навпаки можливо треба розвиватись у цьому напрямку.

1.2 Огляд ігор та ігрових платформ

У наш час існує велика кількість ігрових платформ на кожен гаманець та смак. Та ще більше ігор на кожній з платформ. Заради кращого розуміння у цьому розділі представлені класифікації ігор та ігрових платформ за багатьма ознаками та параметрами.

1.2.1 Категорії

Кожна платформа для продажу та дистрибуції ігор має функцію пошуку ігор за їх категорією. Загалом ці категорії пересікаються та є однаковими майже на усіх платформах. Гра може мати декілька категорій. У мобільних магазинах дистрибуції ігор є приблизно 13 категорій ігор:

Action (з англ. - «Дія»). Ігри наповнені великою кількістю подій та потребують активної участі гравця у ігровому процесі. Одна з найширших категорій, що не обмежена жанрами. Дейякі з найкращих ігор мобільної платформи є представниками цієї категорії.

Стратегії. Це ігри, у яких невизначеність і часто автономні навички прийняття гравцями рішень мають високе значення при визначенні результату. Практично всі стратегії вимагають внутрішнього мислення і, як правило, дуже високої ситуативної обізнаності. Ця категорія може пиишатись наявністю і її рядах таких шедеврів як: «Plague Inc.», «FNAF», «Mini metro» та звичайно легендарна серія «Worms».

Спорт. Категорія спортивних ігор щільно зв’язний з такими жанрами як «симулятори» та «стратегії». Це ігри у яких симулюється гра у спорт. Майже усі існуючі спортивні напрмки мають ігри на їх основі (командні спорти, польові, екстримальні, бойові мистетцтва). Деякі ігри цієї категорії фокусуються на самому спорті та грі у спорт (наприклад серія «Madden NFL») у той час як інші (наприклад «Championship Manager») фокусуються на стратегічній стороні спортивних реалій (тренерство, менеджмент, сфера загалом). Особливістю спортивних ігор є те, що у них можна зустріти реальні назви та імена спортивних команд, гравців, бреднів та інших. Також спортивні ігри оновлюються набагато частіше, щоб відображати сучасний стан, та склад команд у всіті. Це одна з найстаріших та найпопулярніших категорій.

Слова. Категорія ігор у основі яких слова. Кросворди, загадки, та часткова доля

«навчальної» категорії - все це словесні ігри.

Симулятори. Як і актегорія «екшн» ігор - одна з найбільших категорій оскільки «симуляцією» фактично може бути що завгодно. Категорія не обмежена жодним з жанрів та активно використовується всюди.

Сімейні. Для категорії сімейних ігор характерна наявність режимів для декількох користувачів одночасно.

Рольові. Рольова гра («role-playing game» або скорочено «RPG») - це гра, в якій гравці беруть на себе роль персонажів у вигаданій обстановці. Гравці беруть на себе відповідальність за виконання цих ролей в рамках розповіді, або через буквальну дію, або через процес структурованого прийняття рішень щодо розвитку персонажа. Дії, здійснені в рамках багатьох ігор, приводять до успіху або невдачі за формальною системою правил та вказівок. Чудовим прикладом категоріх є шедевральна «Papers, Please».

Пригоди. Пригодницька гра - це відеоігра, в якій гравець бере на себе роль головного героя в інтерактивній історії, керованій дослідженнями та розгаданням головоломок. Зосередженість жанру на сюжеті дозволяє йому значною мірою черпати з інших засобів масової інформації, літератури та фільму, що базуються на оповіді, охоплюючи широке різноманіття літературних жанрів. Багато пригодницьких ігор (текстових та графічних) розраховані на одного гравця, оскільки такий акцент на сюжеті та персонажі ускладнює багатокористувацький дизайн.

Настільні. Настільна категорія ігор включає у собі в основному портовані версії існуючих настільних ігор (шахи, шашки, монополія та інші) та оригінальні настільні мобільні ігри. Для цієї категорії характерна повільність геймплею як і для стратегій.

Музика. Одна я найскладніших та в одночас найкреативніших категорій. Ігри у ядрі ігрового процесу яких знаходиться музика. Світ ігор цієї категорії реагує на музику та спонукає гравця. Гарний представник категорії - музикальний платформур «Geometry Dash».

Карткові. Категорія щільно пов’язана з категорією Казино. Загалом - це портовані версії різних існуючих світових карткових ігор (наприклад наших «Дурня»), або ігри з інших категорій однією з ключових механік яких є гра у карти.

Представники категорії - «Reigns», «UNO», «Hearthstone».

Казино. Ті ж карткові ігри (та не тільки) але з поглибленням у сторону азартних ігор.

Казуальні. Ігри орієнтовані на широку, масс-саркет аудиторію, навідміну від більш «хардкорних» ігор. Можуть бути представниками будь-якої з категорій, є «загальною» категорією. Для казуальних ігор характерна наявність простих правил, значно коротших ігрових сесій, та вони не вимагають від гравця наявності особливих навичок чи знань.Що робить категорію відкритою для найширшої аудиторії гравців.

1.2.2 Рушії

Кожна гра розроблена за допомогою якогось ігрового рушія (інструмента завдяки якому розробним може втілювати ігри).Їх існує велика кількість та всі вони різні, зі своїми плюсами, мінусами та особливостями. Вибір правильного рушія є основою розробки мобільної гри. Вибір роблять на основі підтримки рушієм потрібних платформ, мов програмування, та більш унікальних інших особливостей.

Cocos2d. Випуск - 2008 рік ,Open-Source . Бесплатний. Мови - LUA, C++, JavaScript . Спеціально розробений для 2d,простий інтерфейс, велика спільнота розробників, бібліотека навчальних матеріалів. Використовувався для розвобки таких хітів як: «Family Guy», «The quest for stuff», «Geometry dash », та «Bad land».

Buildbox. Випущений у 2015 році, компанією 8CELL INC. Ціна - від $8 - $99 на місяць. Не потребує знань мов програмування. Є дуже швидким інструментом для розробки, має зручний інтерфейс, ідеальний варіант для розробки «гіперказуальних ігор». Має 20000 превстановлених ассетів (картинок та анімацій) для користування. Ідеальний для початківців. Незважаючи на новизну - приблизно 150 ігор зроблених на цьому рушії вже займають місця у топ-чартах AppStore.

Godot. Випущений у 2018 році, бесплатний, підтримує такі мови як: C#, C++, CGSCRIPT. Godot є ще одним гарним рішенням кросплатформенного рушія для розробки 2D та 3D ігор. Вся виручка від ігор зроблений на цьому рушії йде беспосередньо розробнику.

GDEVELOP. Випущений у 2015 році, бесплатний, не потребує знань мов програмування. Має інтерфейс візуального програмування, спеціалізується на 2D

іграх, єкспорт ігор у одне натискання. Вся виручка від ігор зроблений на цьому рушії йде беспосередньо розробнику.

Unity. Випущений у 2005-му році компанією UNITY TECHNOLOGIES, ціна - від бесплатної версї до $125 на місяць. Мови - C#. Вище перечислені менші ігрові двигуни і нові ігрові двигуни, але не будемо забувати гігантів індустрії, одне з найвідоміших імен у цьому списку - це Unity. Рушій вибору для Nintendo, коли справа дойшла до портування Маріо на мобільні пристрої із запуском Super Mario. Інші мобільні ігри, зроблені в Unity, включають «reigns», і чудова головоломка «Monument Valley». Unity є одним з найбільш універсальних двигунів, з яким розробники мобільних ігор можуть працювати майже не маючи обмежень. Відповідно до сайту Unity - приголомшливі 34 відсотки перших 1000 безкоштовних мобільних ігор, зроблені за допомогою цього рушія. Деякі з цих назв включають «fruit bump », «pirate kings», та «demolition derby». Щодо підтримуюмих платформ. Легше назвати платформи які Unity не підтримує. Підтримуються більш ніж 25 платформ, включаючи iOS, Android, Nintendo Switch, Xbox One і PS4. Універсальний вибір, що є справжнім «молотом» у порівнянні зі попередніми «скальпелями».

Unreal Engine. Випущений у 1998-му році компанією Epic Games, бесплатний, але 5% виручки (якщо вируска більше $3000) іде компаніх Epic Games. Мови - C++. Один з старіших ігрових рушіїв, має усі можливі інструменти, можна створювати кат-сцени з його використанням. З його створенням, що починається з 1998 року, Unreal Engine є найстарішим і повинен бути одним з найвідоміших рушіїв розробки ігор у цьому списку. Це рушій, розроблений компанією Epic Games, назва якої далеко не реальна гіпербола. Unreal Engine був розроблений для шутерів від першої особи, але з тих пір вони використовуються для створення ігор у різних жанрах. Unreal Engine включає в себе повний набір інструментів для розробки. Unreal Engine може все зробити, але, можливо, найбільш унікальною особливістю Unreal Engine є у нього є всі інструменти, необхідні для створення власних найсучасніших інструментів для кінематографічних інструментів, створених для Unreal. Нерухомість професіоналів з кіно та телебачення Unreal Engine можна експортувати на платформи, включаючи iOS, Android, Linux, Xbox, Nintendo Switch, ps4 і більше. Такі успішні

релізи як «Injusctice», «PUBG», - і звичайно, «Fortnite» - усі вони були розроблені на Unreal Engine. Кросплатформна стратегія включала мобільні платформи також. З точки зору ціни Unreal Engine 4 можна безкоштовно використовувати, але після релізу вашої гриEpic Games, мають право на 5% від усіх надходжень після перших трьох тисяч доларів доходу.

SpriteKit. Випущений компанією Apple у 2014-му році, бесплатний, підтримує такі мови як Swift та Obj-c. Є рідним рушієм платформи iOS, достатньо простий у вивченні, має чудову документацію, може бути ідеально вбудований як частина будь-якого не ігрового доатку під платформу iOS. Спеціалізується на 2D.

1.2.3 Операційні системи

Ігри можна також класифікувати за підтривуваними операційними системами. З точки зору розробника для збільшення виручки та клієнтської бази користувачів - вигідно рідтримувати як можливо більшу кількість операційних систем. Питання у тому чи варто це робити ? Чи принесе підтримка , наприклад менш популярної системи, бажаний приріст користувачів чи ні ? Тому загалом підтримують обмеену кількість операційних систем, що формують основний перечень ОС , що є перспективними або навіть обов'язковими для максимальної доступності гри користувачам. Підтримувані ОС можуть бути трьох форм-факторів: комп'ютерні, консольні, мобільні. До мобільних ОС належать основні дві - iOS та. Android. Також існує багато менш популярних та вузько направлених мобільний ОС, але загалом вони не є релевантними у питанні ігор. До комп'ютерних належать Windows, Linux та MacOS. До консольних належать Xbox OS, Orbis OS (PlaystationOS), NintendoOS.

iOS (раніше iPhone OS) - це мобільна операційна система, створена та розроблена компанією Apple Inc. виключно для її обладнання. Саме операційна система в даний час забезпечує багато мобільних пристроїв компанії, включаючи iPhone і iPod Touch. Пристрої iPad до введення iPadOS у 2019 році також працювали на цій ОС. Це друга за популярністю мобільна операційна система в усьому світі після Android. Це основа для інших операційних систем компанії Apple Inc, таких як iPadOS, tvOS та watchOS.

Android - це мобільна операційна система, заснована на модифікованій версії ядра Linux та іншого програмного забезпечення з відкритим кодом, розробленого

головним чином для мобільних пристроїв із сенсорним екраном, таких як смартфони та планшети. Android розробляється консорціумом розробників, відомим як Альянс відкритих телефонів і комерційно спонсорується Google. Він був оприлюднений у 2007 році, перший комерційний пристрій Android був запущений у вересні 2008 року.

Microsoft Windows, яку зазвичай називають Windows, - це група декількох власницьких сімейств графічних операційних систем, усі вони розроблені та продані Microsoft. Кожна сім'я обслуговує певний сектор обчислювальної галузі. До активних сімей Microsoft Microsoft належать Windows NT та Windows IoT; вони можуть охоплювати підсемейства, наприклад Windows Server або Windows Embedded Compact (Windows CE). До неіснуючих сімей Microsoft Microsoft належать Windows 9x, Windows Mobile та Windows Phone.

Лінукс (англ. *Linux,* повна назва — GNU/Linux) — загальна назва UNIX-подібних операційних систем на основі однойменного ядра. Це один із найвидатніших прикладів розробки вільного (free) та відкритого (з відкритим кодом, open source) програмного забезпечення (software). На відміну від власницьких операційних систем (на кшталт Microsoft Windows та MacOS X), їхні вихідні коди доступні всім для використання, зміни та поширення абсолютно вільно (в тому числі безкоштовно).

MacOS - POSIX-сумісна операційна система корпорації Apple. Є спадкоємицею Mac OS 9 — так званого *остаточного релізу* «класичної» Mac OS — основної операційної системи корпорації Apple з 1984 року. OS X входить в сімейство операційних систем Apple OS X, до якого також належить ОС для мобільних пристроїв — Apple iOS. У macOS використовується ядро Darwin, засноване на мікроядрі Mach, що містить код, написаний самою компанією Apple та код, отриманий з ОС NeXTSTEP та FreeBSD. Apple macOS випускається для комп'ютерів Macintosh (Макінтош) на базі процесорів PowerPC та Intel (починаючи з версії 10.6,) macOS підтримує тільки комп'ютери Mac на базі процесора Intel. Mac OS — друга за популярністю у світі операційна система.

Системне програмне забезпечення **Xbox** One, яке іноді називають ОС Xbox або Dashboard Xbox (коли людина має на увазі оновлення програмного забезпечення), - це операційна система, розроблена виключно для консолей Xbox One. Це операційна

система на базі Microsoft Windows, що використовує монітор віртуальної машини Hyper-V і містить окремі операційні системи для ігор та додатків, які можуть працювати на консолі. Він розміщений на внутрішньому жорсткому диску для щоденного використання, а також дублюється на внутрішньому NAND-накопичувачі консолі для цілей відновлення та функціонального встановлення на заводі.

Власною операційною системою PlayStation 4 є **Orbis OS**, що є роздрібною версією FreeBSD версії 9.0, яка вийшла 12 січня 2012 року.

Системне програмне забезпечення **Nintendo** Switch - це оновлена вбудована програма та операційна система, що використовується консоллю відеоігри Nintendo Switch. Основна його частина - це початковий екран, що складається з верхньої панелі, програми перегляду скріншотів ("Альбом") та ярликів до електронного магазину, новин та налаштувань Nintendo. Код заснований на різних частинах Android, а також на основі програмного забезпечення Nintendo 3DS.

1.2.4 Режими

Також ігри можна класифікувати за присутніми режимами або ж за використанням чи відсутністю мереж (доступу до мережі інтернет). Якщо ще років 5 тому назад більшість ігор завжди були оффлайн іграми, тобто не потребували підключення до інтернету - то зараз ринок розрісся та все частіше можна зустріти серйозні продукти , що постійно використовують інтернет для гри , або навіть є повністю онлайн.

Що стосується режимів і іграх - то можна виділити три основних: одиночний, кооперативний та звичайно мультиплеейний. У той час. Як одиночний режиму характерна відсутність присутності інтернету - кооперативний та мультиплеєрні режими у іграх є суто інтернет режимами гри. Незважаючи на це, є комбіновані рішення, та іноді навіть одиночні ігри потребують наявність постійної присутності інтернету (робиться це для як частина security-рівня додатку, або просто для відслідковування деяких даних чи параметрів додатку, або для обох причин обночасно).

1.2.5 Ігрові платформи

Платформи, за допомогою яких грають у відеоігри, розвиваються з самого початку. Від простих ігрових автоматів у залах кінотеатрів (пінбол та інш.) до аркадних машин та комп. Клубів. Можна виділити 3 основні, з однією «підплатформою». А саме: ПК, мобайл, консолі та портативн консолі.

Персональні комп'ютери (**ПК**) - це одна із найзручніших платформ для відео ігор, доступних сьогодні. Це можуть бути як настільні ПК так і ноутбуки. Ринок переповнений комп'ютерами які пібійдуть будь-якому користувачеві. Обладнання повинно включати потужну оперативну пам'ять, а також відеокарту для обробки даних, відому як GPU. Потужні GPU вважаються кращими для отримання чудового результату. Разом з цими та іншими поширеними апаратними засобами ПК, такими як монітор, клавіатури та миші, можна легко грати. Персональні комп'ютери також можна налаштувати за допомогою додаткових пристроїв, таких як джойстики, щоб покращити роботу. Цей досвід також можна покращити, підключивши комп'ютери до великих екранів телевізора за допомогою кабелів типу HDMI або VGA, які підтримує телевізор. Разом з усім цим персональні комп'ютери служать чудовою ігровою платформою.

Консолі для відеоігор - це пристрої, створені спеціально для гри у відеоігри. Зазвичай вони постачаються з пристроями введення, такими як джойстик та основний блок, який виконує всі процеси обробки. Зазвичай їх підключають до екранів телевізорів чи моніторів, щоб побачити візуальний зворотний зв'язок від консолей. Сьогодні на ринку є кілька основних типів консолей. Популярними є, наприклад, **Xbox** (до платформи xbox відносяться всі консольні продукти компанії Microsoft, а саме: Xbox, Xbox 360, Xbox One, Xbox one S, Xbox one X, та Xbox Series X. Консолі платформи Xbox є більш популярними у США та америці у порівнянні з Європою та Азією) , **Wii** та **PlayStation** (до платформи playstation відносяться всі консольні продукти компанії Sony, а саме: Playstation 1, Playstation 2, Playstation 3, Playstation 3 Slim, Playstation 4, Playstation 4 Slim, Playstation 4 Pro, Playstation 5, Playstation Portable, Playstation Vita.) . Також є портативні консолі, такі як **Nintendo Switch** (до платформи Nintendo відностяься всі консольні продукти компанії, а саме: Game&Watch, Game Boy, Game Boy Color, Virtual Boy, Nintendo DS, Wii U, Nintendo

Wii, Nintendo Switch, Game Boy Advance, Nintendo 3DS) . Ці портативні приставки мають невеликі розміри та мають власний дисплей. Отже, консолі відеоігор - це ще одна популярна платформа, яка широко доступна в наші дні.

Мобільна (До мобільної платформи належать всі мобільні девайси незалежно від виробника. Це телефони, планшети, смарт-годинники основних сімейств iOS та Android та інших менш популярних мобільних операційних систем)

1.2.6 Ігрові платформи за доступністю

Доступність ігрової платформи є надвичайно важливим фактором для користувача при виборі платформи. До поняття «доступності платформи» входять декілька факторів, а саме: ціна, повноцінність (або самостійність / завершенність платформи) платформи та простота у використанні. Ціна - повністю очевидно зрозуміле поняття. Кожен пристрій має свою ціну. Цей фактор для багатьох є найважливішим при вирішенні яку платформу обрати. Завершеність платформи. Чи треба після покупки ігрової платформи докуповувати ще якусь техніку або програмне забезпечення ? Або можна користуватись платформою як незалежною смостійною одиницею ? Це також важливий фактор у виборі. Не кожен користувач захоче ускладнювати для себе процес здобуття ігрової платформи докупівлею десятка іншої техніки. Простота використання. На початку розвитку персональних комп’терів лише спеціалісти , піонери техніки могли користуватися ПК, для простої домогосподарки командна строка була чимось «іншопланетним». З розвитком Apple це змінилось та їх ПК счинили справжню революцію у світі доступності ПК та розширило цільову аудиторію. С тих пір простота у використанні, або UX є однією з тих речей, що користувачі очікують всюди (від пральної машинки - до користувальницкого інтерфейсу найскладніших із програм). Отже про доступнісьт ігрових платформ далі.

Доступність ПК є комплексним питанням, якщо розглядати доспупність ПК як ігрової платформи - то для пересічного користувача у порівнянні з іншими платформами все дуже складно. Стаціонарний ПК як ігрова платформа навідміну від консолей та тим більше «мобілок» потребує мінімум 2 пристрої вводу інформаціх (миш, клавіатура) та один пристрій виводу (монітор). До того ж не кожен пересічний користувач у змозі зібрати ПК з комплектуючих, що є оптимальним рішенням при використанні ПК як ігрової платформи, якщо користувач хоче не бути обманутим

виорбниками та мати якумога найкращу зборку з точки зору ціна/якість. Ноутбук як ігрова платформа у наші дні річ можлива, але не настільки дешева як , наприклад, консолі, тому також має свої мінуси. Але незважаючи на усі вище перечислені мінуси та складності платформи - з іншої сторони для користувача більш продвинутого та розуміючого свої потреби - це є ідеаьним рішенням. Тим не менш , ціна, навіть при оптимальних збірках всеріно «кусається» (залежачи від рішення, але для хороших результатів у іграх ціна може хитатись від ~ $1000 і не обмежена максимальним значенням).

Доступність Консолей. У порівнянні з ПК тут все значно простіше. Консольний ринок значно менший ніж комп'ютерний та навідміну від ПК ціновий діапазон консолей також менший. Придбати консоль можна за ~ $400 та ціна ця незважаючи на розвиток консолей з роками залишається нерухомою. Навідміну від ПК для користування консолею потрібен тільки один пристрій виводу інформації - монітор або телевізор, пристрій виводу та вводу звуку та вводу інформації ідуть у комплекті, тому так званий підхід «plug and play» так високо цінеться серед консолької платформи. До того ж користувальницький інтерфест та пристрій вводу інформації на консолях значно зрозуміліші пересічному користувачеві. Замість сотен вкладок вікон та кнопок середньостатичний користувальницький інтерфейс консолі обмежений лише тими функціями, що користувач використовую увесь час. А пристрій вводу навідміну від комп'ютерної клавіатури обмежений півтора десятками кнопок, що дозволяє йому бути зручним та ергономчним для людської руки. Доступність консолей як ігрової платформи грає новими фарбами коли мова заходить про портативні консолі. Незначний розмір, зручність та велика кількість ексклюзивних ігор свого часу створила справжній фурор на ринку ігрових консолей з виходом портатисних консолей компанії Sony, а у наші часи Nintendo Switch.

Доступність мобільних пристроїв. Часи коли мобільний пристрій був ознакоюдостатку, а телефон без проводу був розміром з тостер давно пройшли. У наші дні мобільний пристрій є у кожної людини, малий розмір дозволяє переносити телефон у кишені штанів, а потужність флагманских моделей може позмагатися з представниками Пк платформи. Вміст пам'яті деяких моделей порівнюється з зовнішніми жорсткими дисками, а камери телефонів вже витіснили фотоапарати на

лише професійний рівень фотографії. Що стосується питання ціни - ринок мобільних пристроїв може запропонувати пристрій на будь-який розмір гаманця. Мобільні пристрої доступніші ніж коли завгодно. Вони є абсолютно автономними, будь-які зовнішні пристрої навідміну від консолей та ПК є скоріше опціональним акксесуаром, що коліпшує користування аніж обов'язковим рішенням без якого не обійтись.

1.2.7 Ігрові платформи за середою застосування

Ігрові платформи за середою застосування можуть бути двох типів: суто ігрова платформа, або змішана.

До суто ігрових платформ відносяться консолі. Ігри - основна функція заради якої консолі створювались, а лише потім з розвитком сфери розваг функціонал консолей росширився до підтримки інших мультимедіних розваш, як кіно/серіали. Але левова доля застосування консолей - це ігри.

Усі інші ігрові платформи є змішаними з точки зору середи застосування і можливість грати у них є лише однією з функцій.

Мобільна платформа пропонує величезну кількість функцій крім ігрових. Мобільні звінки, велика кількість месенджерів, соціальних мереж, додатки для фото та відео (з зростанням якості камер телефонів , використання фотоапаратури для повсякденного використання не професіоналами майже зникло), фінансів (додатки банків, крипто-гаманці та інші), новин, спорту та здоров'я, навчання, образу життя, музики, медицини, книжки. Та великий ринок додатків , що використовують професіонали своїх напрямків не потребуючи ПК (додатки для графіки та дизайну, додатки для форматування тексту, різноманітні інструменти). Мобільна платформа може запропонувати не тільки ігри та розваги, велика частка ринку мобільних додаткуів також заповнена іншими не менш захоплюючими та потрібними речами.

ПК платформа має найширшу з усіх середу застосування. На відміну від мобільних пристроїв, що незважаючи на стрімкий розвиток все ще є обмеженими у потужності та доступності професійних інструментів, ПК не має жодних кордонів. Найскладніші задачі, що можна уявити - комп'терна платформа має для них інтсрументи. Програмування, монтаж відео/кіно, створення 2/3D граіфки чи моделей, анімація, написання музики. Платформа комп'ютерів може все.

1.3 Основні завдання

Загальним завданням даної дипломної роботи є творення мобільного додатку-гри під платформу iOS (враховуючи усі аспекти розробки додатку-гри, від прорамування, розробки архітектури до втілення графіки, дизайну та інш.) з подальшою публікацією у магазині AppStore.

Завдання:

- Розкриття функціональних особливостей додатку
- Опис інструментальних засобів розробки
- Дослідження архітектури мобільного додатку
- Наведення структури мобільного додатку
- Верифікація результатів дослідження

Висновки до першого розділу

У рамках першого розділу даної дипломної роботи висвітлено теоретичні аспекти мобільної ігрової індустрії, класифіковано та порівняно ігри та ігрові платформи.

Розвиток інтернету та мобільних технологій зробив суттєво вплинув на розвиток мобільні ігри. Мобільний телефон вже не є опціональним гатжетом, чи тільки засобом зв'язку. У сьогоденні це така ж базова річ, що є у кожної сучасної людини, як і електронна пошта. Доступний та дешевий інтернет тільки збільшує кількість потенційних користувачів, а зменшення ціни на телефони знищило всі перешкоди для людей, що не могли придбати мобільні пристрої раніше. Галузь мобільних технологій і мобільних ігор є перспективною як ніколи - тому ,як мінімум треба бути в курсі того що відбувається у світі новітніх технологій, а краще у цому напрямку розвиватись. Саме тому розроюка додатку є актуальним завданням.

Метою даної дипломної роботи є створення мобільного додатку-гри під платформу iOS (у цілях практики, розширення портфоліо виконавця як розробника iOS додатків, та потенційного заробітку у майбутньому

РОЗДІЛ 2
АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ ТА ОПИС РІШЕНЬ

2.1 ОС

iOS (раніше iPhone OS) - це мобільна операційна система, створена та розроблена компанією Apple Inc. виключно для її обладнання. Саме операційна система в даний час забезпечує багато мобільних пристроїв компанії, включаючи iPhone і iPod Touch; він також живив iPad до введення iPadOS у 2019 році. Це друга за популярністю мобільна операційна система в усьому світі після Android. Це основа для інших операційних систем компанії Apple Inc, таких як iPadOS, tvOS та watchOS.

Спочатку iOS був оприлюднений у 2007 році для iPhone першого покоління, iOS з цього часу поширився на підтримку інших пристроїв Apple, таких як iPod Touch (вересень 2007 р.) Та iPad (січень 2010 р.). Станом на березень 2018 року в App Store Apple міститься понад 2,1 мільйона додатків для iOS, 1 мільйон з яких є рідними для iPad. Ці мобільні додатки разом завантажено понад 130 мільярдів разів.

Інтерфейс користувача iOS базується на прямій маніпуляції, використовуючи мультитач-жести. Елементи управління інтерфейсом складаються з повзунків, вимикачів та кнопок. Взаємодія з ОС включає в себе жести, такі як проведіть пальцем, торкніться, пощипуйте і зворотну щіпку, всі вони мають конкретні визначення в контексті операційної системи iOS та його інтерфейсу мультитач. Внутрішні акселерометри використовуються деякими програмами для реагування на струшування пристрою (одним загальним результатом є команда скасування) або обертання його в трьох вимірах (один загальний результат - перемикання між портретним і пейзажним режимом). Apple отримала високу оцінку за те, що вона включила в iOS функції ретельної доступності, що дозволяє користувачам із вадами зору та слуху правильно користуватися її продуктами.

Основні версії iOS виходять щорічно. На всіх останніх пристроях iOS iOS регулярно перевіряє наявність оновлення, і якщо таке доступне, спонукає користувача дозволити його автоматичну установку. Поточна версія iOS 13 була випущена для громадськості 19 вересня 2019 року, де було представлено налаштування користувальницького інтерфейсу та темний режим, а також такі

функції, як перероблений додаток нагадувань, проведіть пальцем клавіатуру та вдосконалений додаток Photos. iOS 13 не підтримує пристрої, які мають менше 2 ГБ оперативної пам’яті, включаючи iPhone 5s, iPod Touch (шосте покоління) та iPhone 6 та iPhone 6 Plus, які досі становлять понад 10% усіх пристроїв iOS. iOS 13 призначений виключно для iPhone та iPod touch, оскільки варіант iPad тепер називається iPadOS.

Отже мною було обрано оперативну систему iOS у якості ОС під яку буде розроблено дипломний додаток.

2.2 Мова програмування

Swift - це багатопарадигма загального призначення, складена мова програмування, розроблена Apple Inc. для iOS, iPadOS, macOS, watchOS, tvOS, Linux та z / OS. Swift розроблений для роботи з рамками Apple Cocoa та Cocoa Touch та великою частиною існуючого коду Objective-C, написаного для продуктів Apple. Він побудований на основі компілятора LLVM з відкритим кодом та включений у Xcode з версії 6, випущеної у 2014 році. На платформах Apple використовується бібліотека часу виконання Objective-C, яка дозволяє C, Objective-C, C ++ та Swift код для запуску в рамках однієї програми.

Apple задумала Swift підтримати багато основних концепцій, пов'язаних з Objective-C, зокрема динамічну диспетчеризацію, широко розповсюджене прив'язування, розширювальне програмування та подібні функції, але "більш безпечним" способом, що полегшує ловлення програмних помилок; У Swift є функції, що стосуються деяких поширених помилок програмування, таких як перенаправлення нульових покажчиків і забезпечує синтаксичний цукор, щоб уникнути піраміди приреченості. Swift підтримує концепцію розширюваності протоколу, системи розширюваності, яка може бути застосована до типів, структурам та класам, яку Apple рекламує як реальну зміну парадигм програмування, які вони називають "протокольованим програмуванням" (подібно до ознак).

Свіфт був представлений на Всесвітній конференції розробників Apple (WWDC). Під час оновлення до версії 1.2 протягом 2014 року та більш серйозного оновлення до Swift 2 на WWDC 2015. Спочатку власницькою мовою версія 2.2 була зроблена з програмним забезпеченням з відкритим кодом під ліцензією Apache 2.0 З

грудня 2015 року для платформ Apple і Linux .

Через версію 3.0 синтаксис Swift зазнав значної еволюції, при цьому основна команда зробила стабільність джерела фокусом у більш пізніх версіях. У першому кварталі 2018 року Свіфт перевершив «Ціль-С» за розміром популярності.

Swift 4.0, випущений у 2017 році, вніс кілька змін у деякі вбудовані класи та структури. Код, написаний з попередніми версіями Swift, можна оновити, використовуючи функцію міграції, вбудовану в Xcode. Swift 5, випущений в березні 2019 року, представив стабільний бінарний інтерфейс на платформах Apple, що дозволило виконувати час роботи Swift в операційні системи Apple. Він сумісний із Swift 4.

Swift 5.1 був офіційно випущений у вересні 2019 року. Swift 5.1 ґрунтується на попередній версії Swift 5 шляхом розширення стабільних особливостей мови до часу збирання із впровадженням стабільності модуля. Впровадження стабільності модуля дозволяє створювати та обмінюватися бінарними рамками, які працюватимуть із майбутніми випусками Swift.

Основні функції та особливості мови Swift. Swift - це альтернатива мові Objective-C, яка використовує сучасні концепції теорії мови програмування та прагне представити більш простий синтаксис. Під час його введення він був описаний просто як "Об'єктив-С без багажу С".

За замовчуванням Swift не виставляє покажчиків та інших небезпечних аксесуарів, на відміну від Objective-C, який широко використовує вказівники для позначення екземплярів об'єкта. Крім того, використання синтаксису Smalltalk-синтаксису Objective-C для здійснення викликів методів було замінено на стиль точкового позначення та систему простору імен, більш звичну програмістам з інших поширених об'єктно-орієнтованих мов (ОО), таких як Java або C #. Swift вводить ідентичні параметри імені та зберігає ключові поняття Objective-C, включаючи протоколи, закриття та категорії, часто замінюючи колишній синтаксис чистішими версіями та дозволяючи цим поняттям застосовуватись до інших мовних структур, наприклад перелічених типів (переліків)

Swift підтримує **закриття** (**closures**) (відомі як лямбди на інших мовах).

Пітримка **рядків**. У середовищі Сосоа та какао сенсор багато загальних класів

були частиною бібліотеки фонду Kit. Сюди входили бібліотеки рядків NSString (за допомогою Unicode), класи колекції NSArray та NSDictionary та інші. Objective-C надав різні шматочки синтаксичного цукру, щоб дозволити створювати деякі з цих об'єктів під час мовлення в межах мови, але колись створені, об'єктами були маніпульовані виклики об’єктів. У Swift багато з цих основних типів були перенесені до ядра мови, і ними можна керувати безпосередньо. Наприклад, рядки невидимо пов'язані з NSString (коли імпортується Foundation) і тепер можуть бути об'єднані з оператором +, що дозволяє значно спростити синтаксис.

Контроль доступу. Swift підтримує п’ять рівнів контролю доступу для символів: відкритий, загальнодоступний, внутрішній, файловий та приватний. На відміну від багатьох об'єктно-орієнтованих мов, ці контролі доступу ігнорують ієрархії успадкування: приватне вказує на те, що символ доступний лише у безпосередній області застосування, файлприват вказує на те, що він доступний лише зсередини файлу, внутрішній вказує на те, що він доступний у модулі, що містить, публічний вказує він доступний з будь-якого модуля, а відкритий (лише для класів та їх методів) вказує на те, що клас може бути підкласом поза модулем.

Опліонали та ланцюги викликів . Важливою новою особливістю у Swift є типи опцій, які дозволяють посиланням або значенням працювати таким чином, як загальний шаблон у C, де вказівник може посилатися на значення або може бути нульовим. Це означає, що необов'язкові типи не можуть призвести до помилки з нульовим вказівником; компілятор може переконатися, що це неможливо.

Value типи. У багатьох мовах, орієнтованих на об'єкти, об'єкти представлені внутрішньо у двох частинах. Об'єкт зберігається у вигляді блоку даних, розміщених у купі, тоді як ім'я (або "обробляти") цьому об'єкту представлено покажчиком. Об'єкти передаються між методами, копіюючи значення вказівника, дозволяючи тим самим базовим даним на купі звертатися будь-хто з копією. На відміну від цього, основні типи, такі як цілі числа та значення з плаваючою комою, представлені безпосередньо; ручка містить дані, а не вказівник на них, і ці дані передаються безпосередньо методам шляхом копіювання. Ці стилі доступу називаються прохідними посиланнями у випадку об’єктів та значенням pass-by-value для основних типів

Протоколо-орієнтоване програмування. Ключовою особливістю Objective-C є підтримка категорій, методів, які можна додати для розширення класів під час виконання. Категорії дозволяють розширювати класи на місці для додавання нових функцій без необхідності підкласи або навіть доступу до вихідного вихідного коду. Прикладом може бути додати підтримку перевірки орфографії до базового класу NSString, що означає всі екземпляри NSString у додатку перевірки орфографії. Система також широко використовується як організаційна техніка, що дозволяє збирати відповідний код у бібліотечні розширення. Swift продовжує підтримувати цю концепцію, хоча їх тепер називають розширеннями, і декларується за допомогою розширення ключового слова. На відміну від Objective-C, Swift також може додавати нові властивості аксесуарів, типів та переліків до існуючих екземплярів.

Управління памяттю. Swift використовує автоматичний підрахунок довідок (ARC) для управління пам'яттю. Apple вимагала ручного управління пам’яттю в Objective-C, але представила ARC у 2011 році, щоб полегшити розподіл пам'яті та розібрання місця. Однією з проблем ARC є можливість створення потужного еталонного циклу, коли об'єкти посилаються один на одного таким чином, щоб ви могли дістатися до об'єкта, з якого ви почали, слідуючи посиланнями (наприклад, посилання B, B посилання A). Це змушує їх просочитися в пам'ять, оскільки вони ніколи не випускаються. Swift надає ключові слова слабкими і невідомими для запобігання сильних еталонних циклів. Як правило, стосунки батько-дитина використовували б міцний орієнтир, тоді як дитина-батько використовував би або слабке посилання, де батьки та діти можуть бути неспорідненими, або невідомо, де дитина завжди має батьків, але батько може не мати дитину. Слабкі посилання повинні бути необов'язковими змінними, оскільки вони можуть змінюватися і ставати нульовими.

Замикання в класі також може створити міцний контрольний цикл, фіксуючи власні посилання. Самостійні посилання, що трактуються як слабкі або невідомі, можуть бути вказані за допомогою списку захоплення.

Починаючи з 2014-го року коли мова програмування Swift була випущена вона потроху почала витісняти Objective-C з позиції головної мови для розробки додатків під техніку компанії Apple. На сьогоднішній день мова Swift є основною у iOS

розробці, вона зрозуміліша, простіша, швидша та краща за Objective-C, саме тому вона була обрана мною як мова для розробки дипломного додатку-гри.

2.3 Архітектура додатку

У сучасних реаліях iOS розробки існує чимало архітектурних паттернів та підходів (MVC, MVP, MVVM, Clean Architecture, Undirectional Data Flow, VIPER, VIP, SwiftUI…), кожен з яких має свої плюси та мінуси. У цьому підрозділі буде порівняно та описано найпопулярніші з них та обгрунтовано вибір архітектурного патерну, що буде використовуватись при розробці дипломного додатку.

Відчуваєте себе дивно під час роботи з MVC в iOS? У вас є сумніви щодо переходу на MVVM? Чув про VIPER, але не впевнений, чи воно варто того?

Ви збираєтеся структурувати свої знання про архітектурні зразки в середовищі iOS. Ми коротко розглянемо деякі популярні та порівняємо їх у теорії та практиці, подавши кілька крихітних прикладів. Освоєння шаблонів дизайну може викликати звикання, тому будьте обережні: ви можете в кінцевому підсумку задавати собі більше питань, ніж раніше, як-от такі:

Хто повинен мати власний запит на мережу: модель чи контролер?

Як передати модель у модель перегляду нового перегляду?

Хто створює новий модуль VIPER: Маршрутизатор або Презентатор?

Навіщо дбати про вибір архітектури? Тому що, якщо ви одного разу не налагодите величезний клас із десятками різних речей, ви виявите, що не можете знайти та виправити помилки у своєму класі. " Зрозуміло, важко пам’ятати про цей клас як про цілі сутності, таким чином, вам завжди бракує важливих деталей. Якщо ви вже в цій ситуації зі своєю заявою, дуже ймовірно, що:

- Цей клас є підкласом UIViewController.
- Ваші дані зберігаються безпосередньо в UIViewController
- Ваші UIViews майже нічого не роблять
- Модель - це німа структура даних
- Ваші тестові одиниці нічого не охоплюють

І це може статися, навіть незважаючи на те, що ви дотримуєтесь інструкцій Apple і впроваджуєте шаблон MVC від Apple, тож не відчувайте себе погано. У MVC Apple щось не так, але ми повернемося до цього пізніше.

Давайте визначимо особливості гарної архітектури:

- Збалансований розподіл обов'язків між суб'єктами, які мають чітку роль.
- Тестувабельність зазвичай виходить із першої функції (і не хвилюйтеся: це легко з відповідною архітектурою).
- Простота використання та невисока вартість обслуговування.

Чому розподіл?

Розподіл тримає неабияке навантаження на наш мозок, поки ми намагаємось з'ясувати, як все працює. Якщо ви думаєте, що чим більше ви розвиваєтеся, тим краще ваш мозок адаптується до розуміння складності, то ви маєте рацію. Але ця здатність не масштабується лінійно і досягає обмеження дуже швидко. Таким чином, найпростіший спосіб перемогти складність - це розподіл відповідальності між кількома суб'єктами господарювання за принципом єдиної відповідальності.

Чому тустувабельність?

Зазвичай це не питання для тих, хто вже відчув подяку до одиничних тестів, які не вдалися після додавання нових функцій або внаслідок рефакторингу деяких тонкощів класу. Це означає, що тести врятували цих розробників від пошуку проблем під час виконання, які можуть статися, коли додаток знаходиться на пристрої користувача, а виправлення потребує тижня, щоб дістатися до користувача.

Чому простота використання?

На це не потрібно відповіді, але варто зазначити, що найкращим кодом є код, який ніколи не писався. Тому чим менше у вас коду, тим менше помилок у вас. Це означає, що бажання писати менше коду ніколи не повинно пояснюватися лише лінistю розробника, і ви не повинні надавати перевагу розумнішому рішенню, яке закриває очі на його вартість обслуговування.

MV(X) основи.

На сьогоднішній день у нас є багато варіантів, що стосується моделей дизайну архітектури:

MVC

MVP

MVVM

VIPER

Перші три з них передбачають розміщення об'єктів програми в одній з 3 категорій:

- Моделі - відповідальні за дані домену або рівень доступу до даних, який маніпулює даними, наприклад, класи "Person" або «PersonDataProvider".
- View - відповідальний за рівень презентації (GUI), для iOS-середовища продумуйте все, починаючи з префіксу «UI".
- Controller / Presenter / ViewModel - клей або посередник між Моделею та View, як правило, відповідальний за зміну Моделі, реагуючи на дії користувача, які виконуються на View та оновлення View із змінами в Моделі.

Розподіл сущностей дозволяє нам:

- Розуміти їх краже
- Перевикористовувати їх
- Тестувати їх незалежно одна від одних

Почнемо з розгляду паттернів MV(X) та повернемоя до VIPER пізніше.

MVC. Перед тим як розглянути Apple версію MVC давайте розглянемо класичну імплементацію цього патерну.

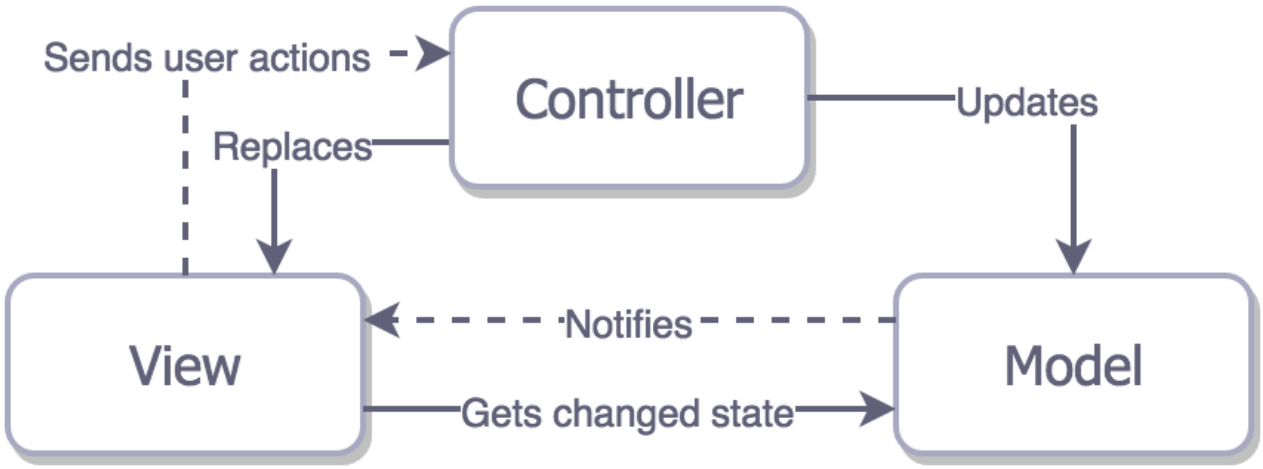


Рисунок 2.1 -

У цьому випадку View є без станів. Він просто надається контролером після зміни Моделі. Подумайте, що веб-сторінка повністю перезавантажена, як тільки ви

натиснете на посилання, щоб перейти кудись ще. Хоча в програмі iOS можливо реалізувати традиційний MVC, це не має особливого сенсу через архітектурну проблему - всі три об'єкти щільно пов'язані між собою, кожен об'єкт знає про інші два. Це різко знижує багаторазове використання кожного з них - це не те, що ви хочете мати у своїй програмі. З цієї причини ми пропускаємо навіть намагаючись написати канонічний MVC-приклад.

Традиційний MVC не можливо використовувати у iOS розробці.

Apple MVC в ідеалі. Контролер є посередником між View і Моделлю, щоб вони не знали один про одного. Найменше багаторазове використання - це Контролер, і це, як правило, добре для нас, оскільки ми повинні мати місце для всієї тієї хитрої ділової логіки, яка не вписується в модель. Теоретично це виглядає дуже просто, але ти відчуваєш, що щось не так, правда? Ви навіть чули, як люди, що скарочують MVC, як контролер масивного перегляду. Крім того, розробка контролера перегляду стала важливою темою для розробників iOS. Чому це відбувається, якщо Apple просто взяла традиційний MVC і трохи покращила його?

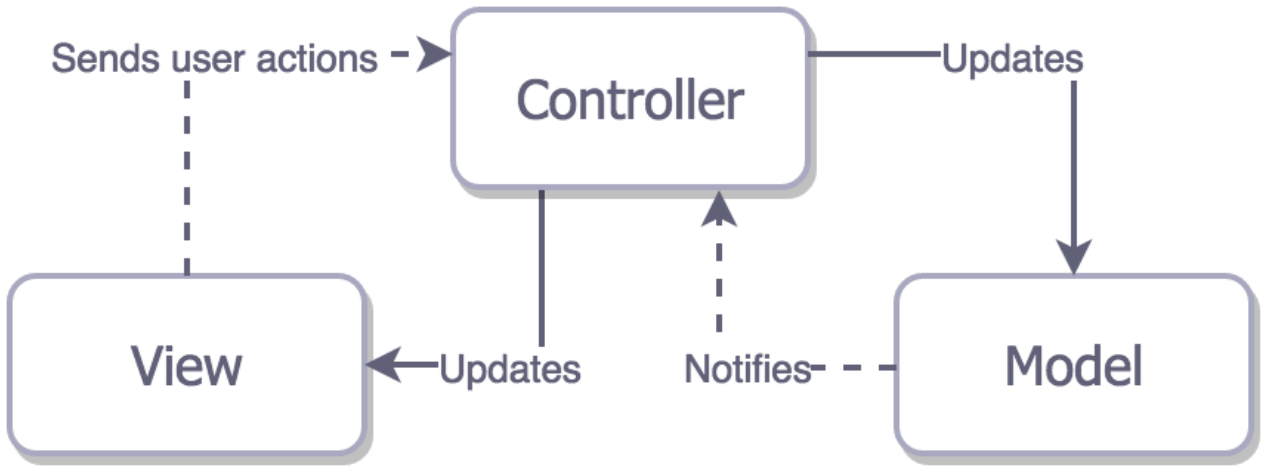


Рис. 2.2 – MVC в ідеалі

Apple MVC у реальності.

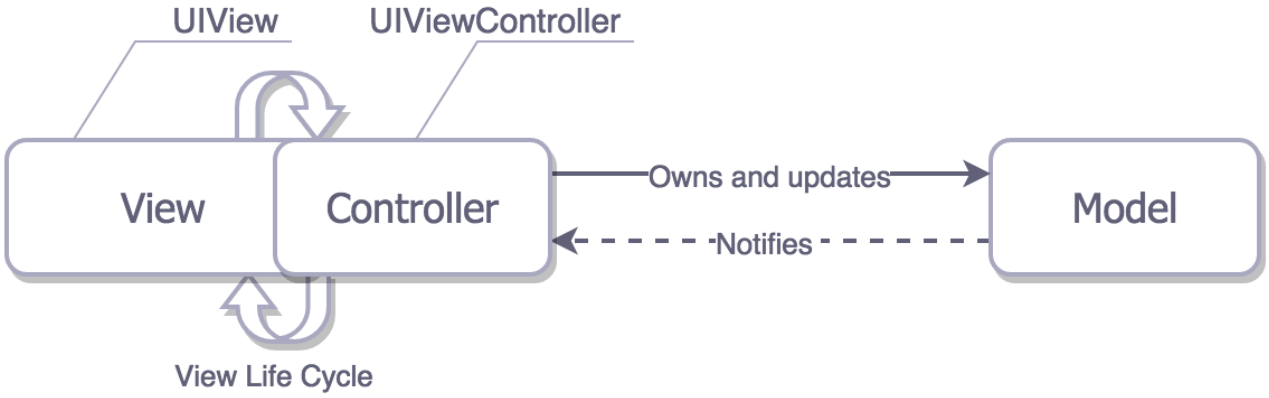


Рис. 2.3 – MVC в реальності

Паттерн MVC у класичній Cocoa розробці примушує наш писати «МасивніViewController'и», оскільки вони настільки втягнуті в життєвий цикл View, що важко сказати, що вони окремі. Хоча ви все ще маєте змогу вивантажити частину ділової логіки та перетворення даних на Модель, у вас немає великого вибору, коли справа стосується перевантаження роботи на View, у більшості випадків вся відповідальність Погляду полягає в надсиланні дій до Контролера. Контролер перегляду в кінцевому підсумку стає делегатом і джерелом даних про все, і, як правило, несе відповідальність за відправлення та скасування мережевих запитів.

Може здатися, що Cocoa MVC – це досить поганий вибір. Але давайте оцінимо це за характеристиками, визначеними на початку статті:

- Розподіл – View і Модель насправді відокремлені, але View і Контролер щільно з'єднані.
- Заповідність – через погану дистрибуцію ви, ймовірно, будете протестувати лише свою модель.
- Простота використання – найменша кількість коду серед інших моделей. Крім того, всі знайомі з цим, таким чином, це легко підтримується навіть недосвідченими розробниками.

Cocoa MVC – це обраний вами шаблон, якщо ви не готові вкладати більше часу в свою архітектуру, і відчуваєте, що щось з більш високими витратами на обслуговування – це надмір для вашого крихітного домашнього вихованця.

Cocoa MVC – найкращий архітектурний зразок з точки зору швидкості

розробки.

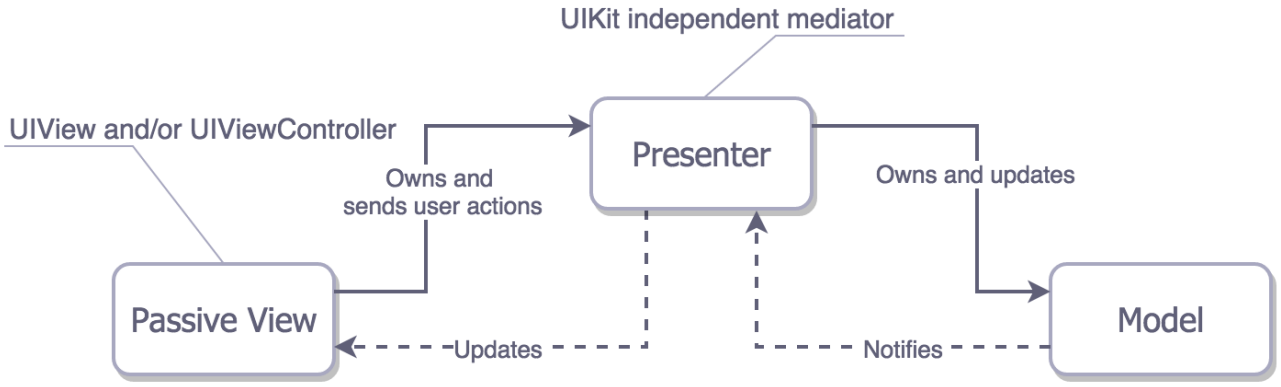


Рис. 2.4 - MVP

MVP. «Мінуси Cocoa MVC втілені».

Чи не схоже це на MVC Apple? Так, так і називається MVP (варіант "Пасивний View"). Але зачекайте хвилину ... Це означає, що MVC Apple насправді є MVP? Ні, це не так, тому що, якщо ви пам’ятаєте, там перегляд тісно поєднаний з контролером, тоді як посередник MVP, ведучий, не має нічого спільного з життєвим циклом ViewController, і з виду можна легко знущатися, коду макета в Презентаторі взагалі немає, але він відповідає за оновлення View даних і стану.

З точки зору MVP, підкласи UIViewController - це фактично View, а не Presenter. Ця відмінність забезпечує чудову перевірку, яка виходить за рахунок швидкості розробки, тому що ви повинні зробити вручну дані та події прив’язуючими.

Важлива примутки. MVP - це перша модель, яка розкриває проблему складання, яка виникає через наявність трьох фактично окремих шарів. Оскільки ми не хочемо, щоб View знав про модель, це неправильно виконувати збірку в представленні ViewController (який є Presenter), тому нам доведеться це робити десь в іншому місці. Наприклад, ми можемо створити послугу маршрутизаторів, що відповідає загальному додатку, яка буде відповідати за складання та презентацію "Перегляд у вік" Це питання виникає і має бути вирішене не лише у MVP, але й у всіх наступних моделях.

Давайте розглянемо особливості MVP:

- Розподіл - ми маємо більшість обов'язків, поділених між Presenter та моделлю, з досить тупим видом (у прикладі вище Модель також німа).

- Заповітність - відмінна, ми можемо перевірити більшу частину ділової логіки завдяки тупому View.

- Простота використання - у нашому нереально простому прикладі кількість коду подвоюється порівняно з MVC, але в той же час ідея MVP дуже зрозуміла.

Класичний MVP у iOS означає чудову тестопридатність та багато коду.

MVVM

Найновіший та найкращий з MV(X) патернів.

Таким чином, MVVM є новітньою формою MV(X), сподіваємось, що він з'явився з урахуванням проблем, з якими MV(X) стикався раніше.

Теоретично модель-View-ViewModel виглядає дуже добре. Вид і Модель вже нам знайомі, але також і Посередник, представлений як Модель перегляду.

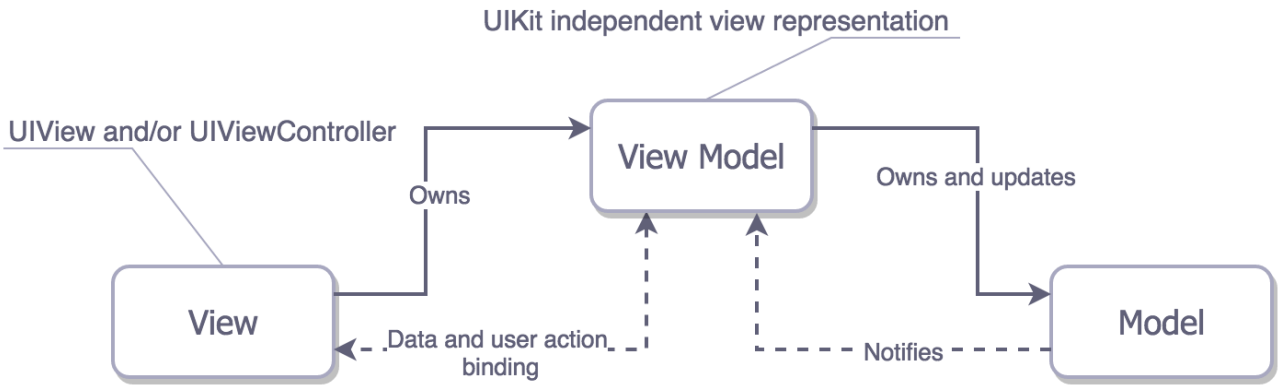


Рис. 2.5 -MVVM

Він досить схожий на MVP:

- MVVM трактує контролер перегляду як Вид
- Між видом та моделлю немає щільної зв'язку

Крім того, він робить обов'язковим, як наглядова версія MVP; однак цього разу не між View і Моделлю, а між View і ViewModel.

Отже, що таке ViewModel в реальності iOS? Це в основному UIKit незалежне представлення Вашого Погляду та його стану. Модель перегляду викликає зміни в Моделі та оновлює себе оновленою моделлю, і оскільки у нас є прив'язка між View та Модель перегляду, перша оновлюється відповідно.

Bindings

Я коротко згадував їх у частині MVP, але давайте трохи поговоримо тут. Bindings виходять із коробки для розробки ОС X, але їх у панелі інструментів iOS немає. Звичайно, у нас є KVO та сповіщення, але вони не такі зручні, як бінди.

Отже, якщо ми не хочемо писати їх самі, у нас є два варіанти:

- Одна з бібліотек зв'язування на базі KVO, наприклад, RZDataBinding або SwiftBond
- Повномасштабні звіри функціонального реактивного програмування, такі як ReactiveCocoa, RxSwift або PromiseKit.

Насправді, сьогодні, якщо ви чуєте "MVVM" - ви думаєте, що ReactiveCocoa, і навпаки. Хоча можна побудувати MVVM за допомогою простих прив’язок, ReactiveCocoa (або побратими) дозволять отримати більшу частину MVVM.

Існує одна гірка правда про реактивні рамки: велика сила приходить з великою відповідальністю. Насправді легко зіпсувати речі, коли ви реагуєте. Іншими словами, якщо ви щось зробите не так, ви можете витратити багато часу на налагодження програми.

Особливості MVVM:

- Розподіл - це не зовсім зрозуміло на нашому крихітному прикладі, але насправді у виду MVVM є більше обов'язків, ніж у MVP View. Оскільки перша оновлює стан свого стану в Моделі перегляду, встановлюючи прив’язки, коли друга просто пересилає всі події до Презентатора і не оновлює себе.
- Заповітність - Модель перегляду нічого не знає про Вид, це дозволяє нам легко перевірити його. Перегляд також може бути перевірений, але оскільки це залежить від UIKit, ви можете його пропустити.
- Простий у використанні - він має таку ж кількість коду, що і MVP у нашому прикладі, але в реальному додатку, де вам доведеться пересилати всі події з представлення до презентатора та оновлювати Перегляд вручну, MVVM буде набагато стрункішим якщо ви використовували прив’язки.

MVVM дуже привабливий, оскільки поєднує в собі переваги вищезгаданих підходів, і, крім того, він не потребує додаткового коду для оновлень View через зв'язки на стороні View. Тим не менш, заповітність все ще на хорошому рівні.

VIPER

Досвід побудови LEGO переданий у дизайн додатків для iOS.

VIPER - наш останній кандидат, що особливо цікаво, оскільки він не походить із категорії MV (X).

На сьогодні ви повинні погодитись, що деталізація відповідальності дуже

хороша. VIPER робить ще одну ітерацію щодо ідеї поділу обов'язків, і цього разу у нас є п’ять шарів.

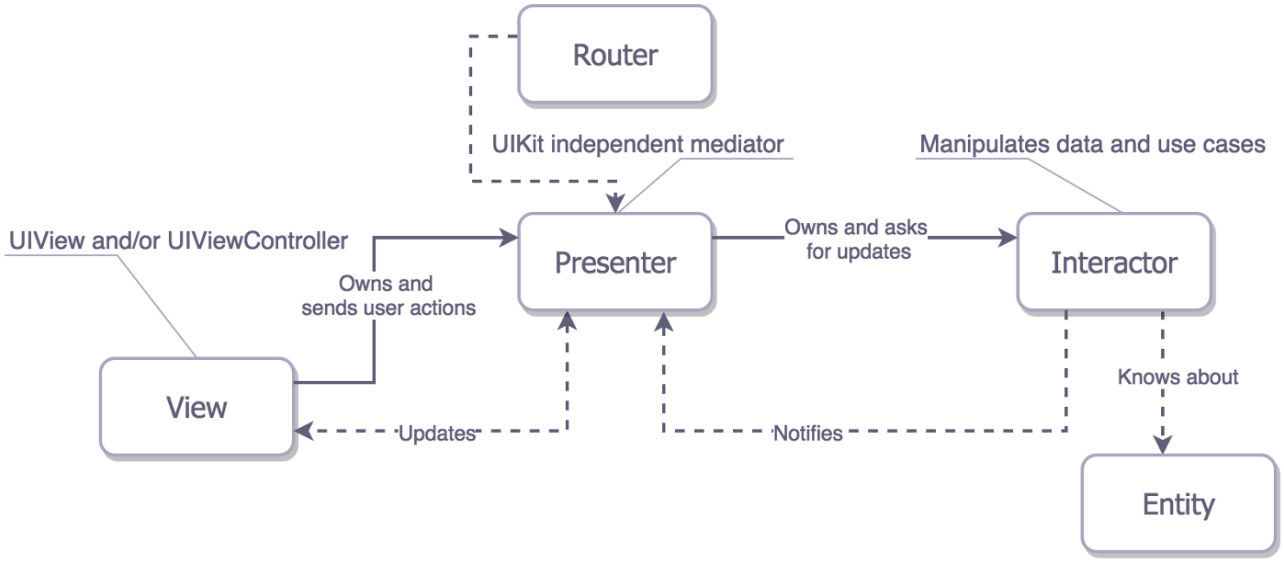


Рис. 2.6 - VIPER

- Інтерактор - містить ділову логіку, пов’язану з даними (Entities) або мережею, наприклад створення нових екземплярів сутностей або отримання їх із сервера. Для цих цілей ви будете використовувати деякі Служби та менеджери, які не вважаються частиною модуля VIPER, а є зовнішньою залежністю.
- Presenter - містить бізнес-логіку, пов'язану з інтерфейсом користувача (але не залежить від UIKit), використовує методи інтерактора.
- Суб’єкти - ваші звичайні об'єкти даних, а не рівень доступу до даних, оскільки це відповідальність Інтерактора.
- Маршрутизатор - відповідає за з'єднання між модулями VIPER.

В основному модуль VIPER може представляти собою один екран або всю історію користувача вашої програми - подумайте про автентифікацію, яка може бути одним екраном або декількома пов'язаними з ними. Наскільки малі бути ваші блоки "LEGO"? - Тобі вирішувати.

Якщо порівняти його з видом MV(X), ми побачимо кілька відмінностей розподілу обов’язків:

- Логіка моделі (взаємодія даних) змістилася в Інтерактор з Сутностями як німі структури даних.

- В Presenter перейшли лише обов'язки щодо представлення користувальницького інтерфейсу Controller / Presenter / ViewModel, але не можливості зміни даних.
- VIPER - це перший зразок, який явно вирішує відповідальність за навігацію, яку повинен вирішити Маршрутизатор.

Правильний спосіб маршрутизації є викликом для iOS-програм, тому що MV (X) моделі просто не вирішують цю проблему.

Особливості MVVM:

- Розподіл - безперечно, VIPER є чемпіоном у розподілі обов'язків.
- Заповітність - тут немає сюрпризів, краща дистрибуція - краща заповідність.
- Простота у використанні - нарешті, два вище коштують ремонту, як ви вже здогадалися. Вам доведеться написати величезну кількість інтерфейсу для занять з дуже маленькими обов'язками.

То стосується «LEGO» ?

Використовуючи VIPER, ви можете відчути, що будуєте Емпайр Стейт Білдінг з блоків LEGO, і це сигнал, що у вас є проблеми. Можливо, зарано приймати VIPER для вашої заявки, і вам слід розглянути щось простіше. Деякі люди ігнорують це і продовжують стріляти з гармати в горобців. Я припускаю, що вони вважають, що їхні додатки отримають вигоду від VIPER хоча б у майбутньому, навіть якщо зараз вартість обслуговування нерозумно висока. Якщо ви вірите в те саме, то рекомендую спробувати Generamba - інструмент для створення скелетів VIPER. Хоча для мене особисто схоже на використання автоматизованої системи націлювання для гармати, а не просто в постріл.

Ми пройшли кілька архітектурних зразків, і я сподіваюся, що ви знайшли відповіді на те, що вас турбувало, але я не сумніваюся, що ви зрозуміли, що немає срібної кулі, тому вибір архітектурного шаблону - це питання вагомих виправ у вашій конкретній ситуації.

Тому природно мати поєднання архітектур в одному додатку. Наприклад: ви почали з MVC, потім зрозуміли, що одному конкретному екрану стало занадто важко підтримувати ефективність із MVC та перейшли на MVVM, але тільки для цього конкретного екрана. Не потрібно переробляти інші екрани, для яких MVC насправді працює добре, оскільки обидві архітектури легко сумісні.

Для розробки додатку мною було обрано патерн MVC тому що для розробки невеликого додатку-гри від підходить найкраще. Модель складається лише з сутності, що відслідковує прогрес гравця. Майже весь процес користування складається з взаємодії з користувальницьким інтерфейсом. Саме тому обрано було MVC.

2.4 UI Архітектура додатку

Навігація

Люди, як правило, не знають про навігацію програми, поки це не відповідає їх очікуванням. Ваше завдання - реалізувати навігацію таким чином, щоб підтримувати структуру та призначення вашого додатка, не привертаючи до себе уваги. Навігація повинна відчувати себе природно і звично, і не повинна домінувати над інтерфейсом і не відводити фокус від вмісту. В iOS є три основні стилі навігації.

Ієрархічна навігація

Зробіть один вибір на екрані, поки не доїдете до пункту призначення. Щоб перейти до іншого пункту призначення, вам слід відстежити свої кроки або почати все спочатку і робити різні варіанти. Налаштування та пошта використовують цей стиль навігації.

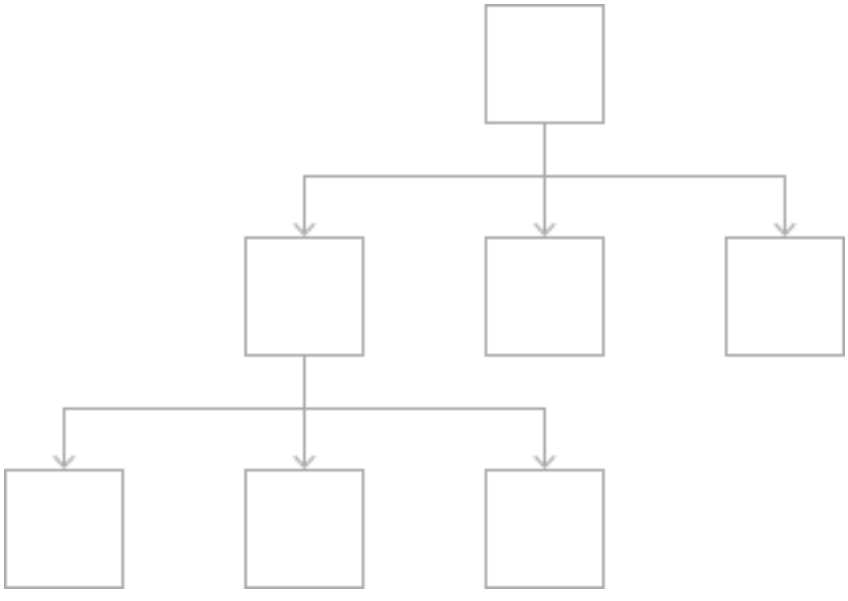


Рис. 2.7 - Ієрархічна навігація

Плоска навігація

Перемикайтеся між кількома категоріями вмісту. Музика та App Store використовують цей стиль навігації.

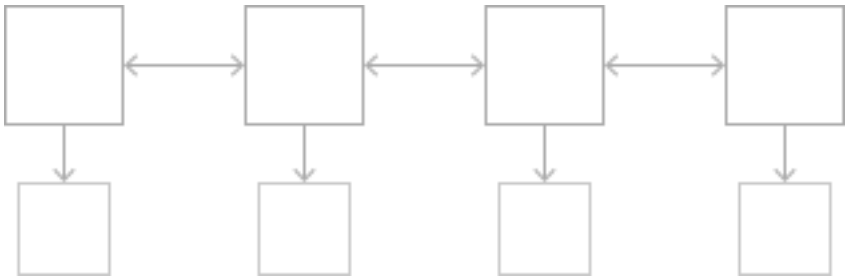


Рис. 2.8 - Плоска навігація

Навігація орієнтована на вміст або досвід

Переміщюйтесь вільно через зміст додатку, або сам контент визначає навігацію. Ігри, книги та інші захоплюючі програми зазвичай використовують цей стиль навігації.

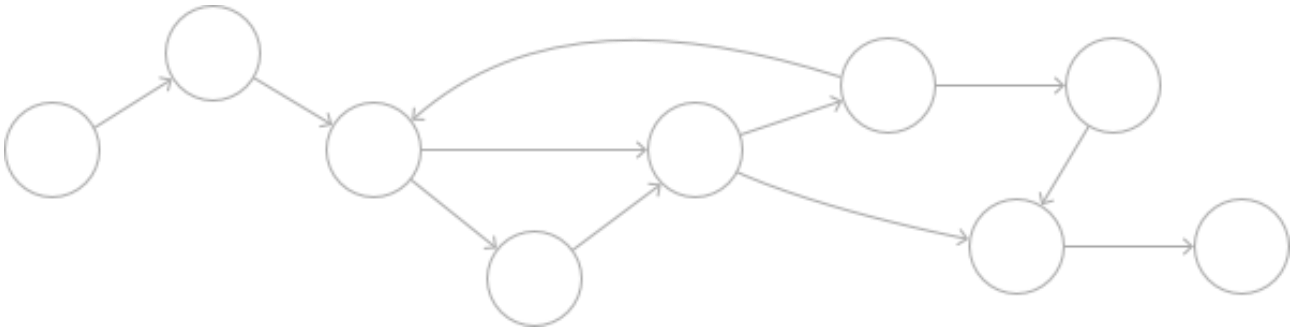


Рис. 2.9 - Досвід навігація

Деякі додатки поєднують у собі кілька стилів навігації. Наприклад, додаток, який використовує рівну навігацію, може реалізовувати ієрархічну навігацію в межах кожної категорії.

Завжди надавайте чіткий шлях. Люди завжди повинні знати, де вони знаходяться у вашому додатку та як дістатися до наступного пункту призначення. Незалежно від стилю навігації, важливо, щоб шлях через вмістбув логічним,

передбачуваним та простим. Загалом, дайте людям один шлях до кожного екрану. Якщо їм потрібно бачити екран у декількох контекстах, спробуйте скористатися аркушем дій, попередженням, переходом або модальним переглядом.

Створіть інформаційну структуру, яка дозволяє швидко та легко дістатись до контенту. Впорядкуйте свою інформаційну структуру таким чином, щоб вимагати мінімальну кількість дотиків, пальців та екранів.

Використовуйте жести дотику, щоб створити плинність. Зручте переміщення по інтерфейсу з мінімальним тертям. Наприклад, ви можете дозволити людям пальцем пальцем з боку екрана, щоб повернутися до попереднього екрана.

Використовуйте стандартні навігаційні компоненти. Коли це можливо, використовуйте стандартні елементи керування навігацією, такі як елементи керування сторінками, панелі вкладок, елементи сегментування, представленнятаблиць, представлення колекцій та розділені види. Користувачі вже знайомі з цими елементами управління та інтуїтивно знають, як обійти ваш додаток.

Використовуйте навігаційну панель для переміщення по ієрархії даних. Заголовок навігаційної панелі може відображати поточну позицію в ієрархії, а кнопка "назад" полегшує повернення до попереднього місця. Докладні вказівки див. У полосах навігації.

Використовуйте панель вкладок, щоб представити однорангові категорії вмісту чи функціональності. Панель вкладок дозволяє людям швидко та легко переходити між категоріями, незалежно від поточного місцезнаходження. Докладні вказівки див. У смугах вкладок.

Використовуйте керування сторінками, коли у вас є кілька сторінок одного типу вмісту. Елемент керування сторінками чітко повідомляє про кількість доступних сторінок і яку зараз активну. Додаток Weather використовує керування сторінками для відображення певних погодних сторінок. Детальні вказівки див. У розділі Керування сторінками.

Інтерфейси у ігрових застосунках

Юзер інтерфейси у ігрових додатках не відрізняються багато чим від інтерфейсів у простих додатках. Розробники користуються тими самимим інструментами при створенні інтерфейсів. Єдина різниця полягяє у тому, що при створенні інтерфейсу гри арсенал інструментів розроника поповнюється інтеракивними ігровими сценами. Навідміну від простих додатків де користування інтерфесом є основним проводженням часу у додатку , у ігрових додатках інтерфейс є провідником від меню до інтерактивних частин ігор, де меню замінюється на інтерфейс управління та взаємодії з грою.

Дизайн інтерфейсу мобільних ігор - це завжди величезна задача. Потрібен такий широкий спектр наборів навичок, щоб перейняти його від концепції до завершення. Крім того, більшість мобільних ігор є їх інтерфейсами, причому переважна більшість ігор відбувається в інтерфейсі, на відміну від багатьох їхніх кузенів ПК та консолей. Щоб зробити це експоненціально складніше, дизайнери інтерфейсів також повинні розуміти монетизацію, утримання та багато інших тем, що стосуються у мобільних іграх.

Існує багато причин, через які проектування інтерфейсу для гри настільки складне, і важливо, щоб ви намагалися зрозуміти всі різні проблеми, з якими ви стикаєтеся чи могли зіткнутися, щоб зробити свою гру успішною. Незалежно від того, чи є ви користувачем інтерфейсу або дизайнером UX, робіть і те, і інше, чи є дизайнером ігор, продуктовим менеджером або будь-якими способами займаєтесь дизайном інтерфейсу гри, важливо пам’ятати про деякі найкращі практики дизайну інтерфейсу. робота.

Є багато важливих аспектів розробки, що важливо не забувати щоб покращити свої ігри... .. інтерфейс і, сподіваємось, зробить вашу гру більш переконливою та успішною. Про аспекти далі.

Використовувати монетизацію - розумійте потреби в монетизації в інтерфейсі. Ви повинні зрозуміти, як ваша гра та інші ігри ефективно (або неефективно використовувати елементи в інтерфейсі, щоб краще монетизувати гру. Деякі з факторів можуть включати: розмір і розміщення кнопок (підкреслюючи кнопки для придбання) речі та мінімізуйте кнопки закриття (наприклад,), світіть або

анімуйте піктограми для спеціальних подій, розпродажів чи інших ключових речей, щоб звернути на них увагу, розмістити важливі функції на видатних місцях, використовувати потрібні кольори або збільшити розмір потрібних кнопок гравці, які торкнуться перших. Є багато хитрощів, які потрібно продумати та використовувати.

Визначте стани - розумійте різні стани інтерфейсу користувача та кожного елемент. Кожен екран і багато кнопок можуть мати безліч можливих різних контекстних станів, в яких він може знаходитися. Наприклад, кнопка "покупка" може бути в одному стані, якщо гравець має можливість (ресурси) придбати предмет, але може змінити стан на " Якщо у програвача не вистачає ресурсів, натисніть кнопку "Купуйте більше ресурсів", яка також натисне інше меню при натисканні. Інші стани можуть бути, якби гравці виконали ряд умов, які дозволили б їм щось зробити. Наприклад, можливо, гравець намагається з'єднати кілька символів нижчого рівня в персонаж вищого рівня, щоб допомогти йому швидше піднятися та покращитися, і кнопка «запобіжник» може не активуватися, поки гравець не введе в меню всі правильні символи. належним чином. У деяких випадках може бути кілька різних елементів, контекст яких змінюється на основі дій гравців. Важливо розуміти різні стани інтерфейсу.

Тестуйте на мобільних пристроях та планшетах . Дуже важливо протестувати свій інтерфейс на кількох пристроях різного розміру. Багато розробників тестують лише на більших планшетах так, як це зручно, і не спроможні протестувати їх на менших телефонах або міні-специфікаціях. Це часто призводить до того, що кнопки занадто малі або шрифти занадто важкі для читання.

Одне із завдань для швидкого тестування макетів - зробити макет потрібного розміру та розмістити його на своєму пристрої та переглянути його як фотографію. Dropbox добре працює для цього. Це простий спосіб побачити, наскільки великі елементи вашого інтерфейсу є на різних екранах.

Використовуйте зображення та графічні репрезентації слів, коли це можливо, та мінімізуйте текст . Тут є дві різні думки. Для ігор я вважаю, що використання більш іконографічного підходу до дизайну інтерфейсів є більш чітким . Використання зображень дозволить показати щось невелике замість багато тексту та може зробити його набагато чистішим для гравців.

Однак у більш сучасних конвенціях щодо дизайну інтерфейсу / UX багато хто зараз стверджує, що для того, щоб зробити інтерфейс інтерфейсу супер швидким і чистішим, слід використовувати весь текст або здебільшого лише текст, щоб зробити інтерфейс значно швидшим та чутливішим. Який би підхід ви не використовували, розумійте плюси та мінуси обох.

Нехай це буде просто і зрозуміло . Зробити простий інтерфейс набагато складніше, ніж складний. Продовжуйте ітерацію в інтерфейсі якомога більше, поки це не стане максимально простим. Це може зайняти багато ітерацій, але простіше. Чим чистіше і простіше його використовувати, тим більше користувачів оцінять це. Спробуйте приховати глибину в більш глибоких меню, якщо це потрібно, щоб основний досвід користувача був максимально простим.

Зробіть його швидким для завантаження . Критично важливо, щоб інтерфейс був максимально чуйним. Гравці ніколи не повинні чекати інтерфейсу. Переконайтеся, що кожен екран, кожна анімація, кожне меню вискакують реагує і миттєво, якщо можливо, не змушуйте гравців чекати.

Використовуйте колір послідовно та змістовно, а не просто, щоб добре виглядало . Дуже важливо правильно використовувати кольори у своєму інтерфейсі для всього, що повинно мати важливу логіку. Використання червоних, зелених та інших стандартних кольорів має певні заздалегідь сприйняті значення для людей, про які ви повинні знати. Колір може бути ВЕЛИЧЕЗНИМ індикатором стану або функції кнопки. Подумайте про такі речі, як, як слід змінити колір зі зміною стану кнопки. Наприклад, коли гравці купують чи продають ресурс, чи колір піктограми та сума однаковий для обох, чи це змінюється, щоб допомогти швидко показати гравцям, чи отримають вони чи втратять ресурс?

Читалємі шрифти - переконайтеся, що ви можете читати шрифти .
Шрифти часто виглядатимуть чудово на планшеті чи на великому екрані, але чи розбірливі вони на маленькому екрані телефону та меншій роздільній здатності? В багатьох Іграх є шрифти, які не може читати більшість людей. Переконайтеся, що ваші шрифти можна читати при будь-якій роздільній здатності та переконайтеся, що у вас є рішення щодо того, як тексти можуть перетікати, змінювати чи прокручувати, щоб зробити їх зрозумілими для гравців.

Розробка базової архітектури інтерфейсу ігрового додатку При розробці архітектури користувальницького інтерфейсу для мого ігрового додатку мною було обрано підхід ієрархічної навігації. При ієрархіній навігації треба робити один вибір на екрані, поки не доїдете до пункту призначення. Щоб перейти до іншого пункту призначення, вам слід відстежити свої кроки або почати все спочатку і робити різні варіанти.

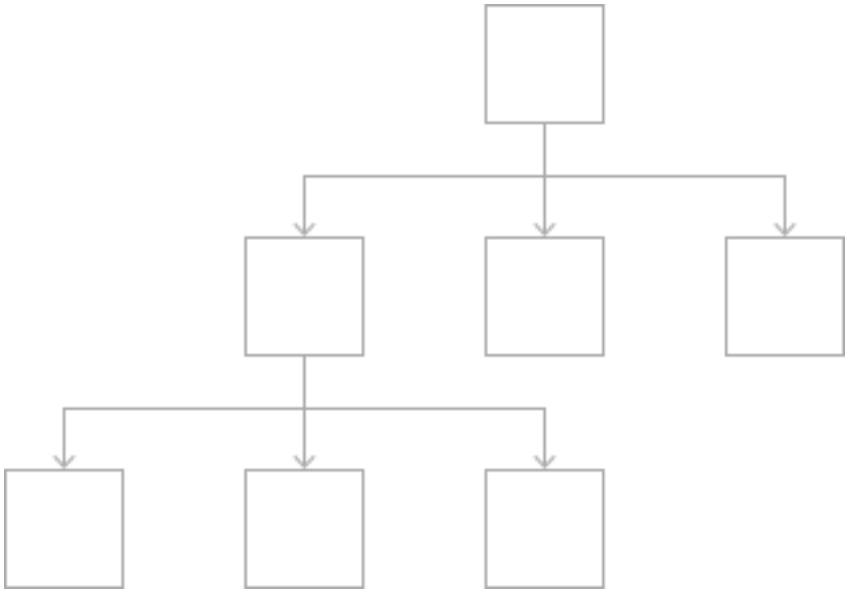


Рис. 2.10 - Ієрархічна навігація

Для ігрових додатків такій тип навігації підходить якнайкраще, бо у ігрових додатках завжди є інтерактивна частина (ігрова сцена) у якій відбувається основна гра і поверх неї побудована вся інша частина додатку. І ця ігрова сцена може бути анімованим фоном основного меню. Таким чином з самого початку запуску додатку користувач вже є заволіченим у гру, та вже починає досліджувати гру.

Щодо візуального способу презентації , мною було обрано спосіб презентації екранів один поверх іншого з ефектом «cross dissolve» (ефект напливу одного контексту поверх іншого).

2.5 Середа розробки

Існує немало інструментів та серед для iOS розробки. Основною та найпопулярнішеє є Xcode. Навідміну від усхі інших рішень, як корсплатформених так і нативних Xcode

випускається Apple тобно безпосередньо розробником самої платформи під фку розроблюються додатки.

Xcode - це інтегроване середовище розробки (IDE) для macOS, що містить набір інструментів для розробки програмного забезпечення, розроблений Apple для розробки програмного забезпечення для macOS, iOS, iPadOS, watchOS та tvOS. Вперше випущений у 2003 році, останній стабільний реліз - версія 11.5 і доступний через магазин додатків Mac безкоштовно для користувачів macOS Catalina. Зареєстровані розробники можуть завантажувати попередні версії та попередні версії пакету через веб-сайт Apple Developer. Xcode включає Інструменти командного рядка (CLT), які дозволяють розвивати стиль UNIX через додаток Terminal в macOS, встановлюючи інструменти для розробників командного рядка.

Набір Xcode включає більшість документації для розробників Apple і вбудований Interface Builder - додаток, що використовується для побудови графічних інтерфейсів користувачів. Найпопулярніший інструмент для побудови інтерфейсів у середі Xcode - це файли типу Storyboard та Xib.

Редактор інтерфейсу Builder в Xcode спрощує проектування повного інтерфейсу користувача без написання коду. Просто перетягніть вікна, кнопки, текстові поля та інші об’єкти на дизайн-полотно, щоб створити функціонуючий інтерфейс користувача.

Оскільки Cocoa та Cocoa Touch побудовані за схемою Model-View-Controller, легко самостійно розробити інтерфейси, окремо від їх реалізації. Користувацькі інтерфейси - це фактично архівовані об’єкти Cocoa або Cocoa Touch (збережені як .nib файли), а macOS та iOS динамічно створюватимуть зв’язок між інтерфейсом користувача та кодом при запуску програми.

Storyboard’и. Повний додаток для iOS складається з декількох view, через які користувач переходить. Взаємозв'язок між цими view визначається за допомогою розкладок, які показують повний вигляд потоку вашої програми користувальницького інтерфейсу розробника інтерфейсу дозволяє легко створювати та розробляти нові представлення даних та з’єднувати їх між собою, щоб створити повний користувальницький інтерфейс, готовий до вашого користувацького коду.

Асистент. Відкрийте помічник під час редагування файлу Storyboard або .xib, щоб швидко підключити елементи управління інтерфейсом до коду, який реалізує їх поведінку. Якщо ви ще не написали код, Xcode запропонує створити заглушку для дії (спосіб запуску) або розетки (змінна для зберігання даних), яка забезпечить логіку вашого інтерфейсу.

Авторозташування. І iOS, і macOS містять потужну систему компонування під назвою Auto Layout, з відмінною підтримкою, вбудованою в Interface Builder. Автоматичний макет заснований на ідеї, що кожен об'єкт у вашому інтерфейсі може визначати обмеження для контролю того, як він реагує на батьківський вигляд та інші елементи управління інтерфейсом. Наприклад, ви можете визначити, чи кнопка залишається певного розміру чи розширюється для розміщення більшого тексту під час відображення іншої мови.

Інтерфейс Builder може автоматично створити для вас всі ваші обмеження, забезпечивши набір сумісних правил. Ви також можете взяти прямий контроль над обмеженнями, щоб визначити точний пріоритет кожного, визначивши, як ваша програма буде працювати в різних розмірах екрана при повороті або при запуску в нових локалях.

Попередній перегляд. Використовуйте режим попереднього перегляду для швидкого перегляду інтерфейсу в різних ситуаціях, не запускаючи додаток, що значно прискорить ітеративний процес проектування. Ви можете переглядати свою програму у портретному чи пейзажному форматі, на попередній версії iOS, у різних розмірах екрана тощо.

Вибір Xcode як середи розробки був повністю очевидним. Це ідеальне рішення для нативної iOS розробки.

2.6 Інструменти для стрворення ассетів

Створення будь-якого додатку потребує великої кількості так званих «ассетів» (картинок, іконок, шпалерів, фонових зображень). Створення додатку-гри у свою чергу потребує ще більшої кількості ассетів (спрайтів перхонажів, текстур, фонів, анімацій). Всі ці асети створюються у якихось програмах, кількість котрих дуже велика, від програм, що створюють анімації скелетів персонажів до ефектів

частинок, такі як дощ, вгонь, дим, іскри… дуже велика від майже універсальних як програми компанії Adobe (такі як Photoshop та інші) до дуже вузьконаправлених як Pixaki.

Дипломний додаток-гра потребує ассети «олдскульного», стилю, як у старомодних іграх коли потужність ігрових платформ та кількість пам'яті не дозволяля обробляти зображення у високій якості. Для розробки ассетів було обрано дві програми: Procreate та Pixaki.

Procreate - це растровий графічний редактор для цифрового живопису, розроблений та опублікований Savage Interactive для iOS та iPadOS. Розроблений у відповідь на мистецькі можливості iPad, він був запущений у App Store (iOS) у 2011 році.

Мета Procreate - відтворити природне відчуття фізичного малювання, використовуючи при цьому практичні переваги цифрової платформи. Він пропонує понад 130 пензлів, безліч шарів, режими накладання, маски, експорт 4K роздільної здатності відео процесів, автоматичне збереження та багато інших цифрових інструментів мистецтва. Окрім растрової графіки, це програмне забезпечення має обмежені можливості редагування та візуалізації тексту та векторної графіки. Procreate розроблений для мультитач і Apple Pencil. Він також підтримує ряд сторонніх стилусів та імпорт / експорт у формат Adobe Photoshop .PSD. Procreate не вимагає покупки через додаток або будь-якої форми підписки. Одними з найважливіших особливостей є підтримка Apple Pencil та простота у шавчанні.

Pixaki. На майбутнє піксельного мистецтва.

Піксельне мистецтво - це набагато більше, ніж просто ретро. Ми вважаємо, що найкраще піксельне мистецтво досі робиться. Вами.

Тому про Pixaki немає нічого ретро. Додаток створено для того, щоб в повній мірі скористатися найновішими функціями iOS, і розроблений для фантастичного досвіду на iPad та iPad Pro.

Піксель над вашими ескізами та фотографіями.

Починати з ескізу або використовувати еталонну фотографію - чудово, але дотепер це створювало певну проблему; якщо змінити розмір зображення, щоб розмістити пікселі вгорі, деталь втрачається, що ускладнює роботу.

Референтні шари Pixaki дозволяють імпортувати будь-яке зображення з вашої бібліотеки фотографій з повною роздільною здатністю, змінити розмір і розмістити його на полотні, а потім намалювати пікселі вгорі. Можна навіть мати декілька еталонних шарів одночасно і відрегулювати непрозорість, щоб ви могли намалювати під зображенням, якщо хочете.

Анімуйте. Мистецтво пікселів було змушене рухатись, і Pixaki із задоволенням втілює ваші твори в життя. З розширеними функціями, такими як тривалість кадрів і шкіра лука, змінена кольором, ніщо не може зупинити вас.

Шари. Створюйте, копіюйте, копіюйте та об'єднуйте шари, підтримуючи до 50 шарів на кадр. Відрегулюйте непрозорість кожного, щоб легко створити ефекти освітлення та напівпрозорості. Упорядкування шарів так само просто, як перетягування.

Повністю підлаштовуються палітри. Pixaki постачається з попередньо встановленими великими палітрами, але також дозволяє створити та зберегти власну. Після збереження палітри ви можете імпортувати її в будь-який документ, над яким працюєте. Ви також можете імпортувати палітру, яку ви створили в іншому програмному забезпеченні або завантажили у формати .aco, .pal та .gpl.

Форми та лінії. Лінійний інструмент дозволяє легко створювати ідеально прямі лінії, які за бажанням можна зафіксувати під кутами, ідеальними для ізометричного мистецтва. Інструменти прямокутника та еліпса допомагають створювати основні фігури, які можна автоматично заповнити кольором, а співвідношення сторін можна заблокувати, щоб створити квадрати та кола.

Потужна заміна кольору. Це може виглядати як старе відро, але це насправді інструмент із заміною кольору. Ви можете використовувати його як звичайний інструмент для відро або замінити всі екземпляри певного кольору іншим. Це можна налаштувати на вплив лише на шар, над яким ви працюєте, на всі шари поточного кадру або всі шари на всіх кадрах. Увімкнення режиму стирання для видалення всіх пікселів заданого кольору. Це прекрасний спосіб працювати з обмеженою палітрою кольорів.

Отже для створення графічних ресурсів додатку обрано програми Procreate та Pixaki.

2.7 Інструменти монетизації

Монетизація мобільних додатків. Є три найбільш популярні моделі - **IAP**, **IAA** та **платні програми**. Покупки через додаток або IAP (In App Purchase) - це обрана модель монетизації для величезної частини розробників ігор на мобільні телефони. Лише **4%** геймерів насправді здійснюють покупки через додаток - але саме вони є тими основними клієнтами, на яких фокусуються розробники freemium ігор. IAP, як правило, включає три різні категорії предметів, які можна придбати: По-перше, **витратні** матеріали/покупки (англ. **Consumables**). Це покупки, які робляться, а потім споживаються у додатку, як-от додаткове життя, додатковий досвід, додаткові ходи або валюта у грі. Зазвичай користувачі купують витратні матеріали для подальшого просування через гру. Користувач також може придбати **не витратні** матеріали (англ. **Non-consumables**), які іноді називаються правами, які часто використовуються щоб розблокувати доступ до функцій або вмісту в грі. Non-consumable IAP можна придбати лише раз і назавжди. Подумайте про додаткові можливості налаштування візулального вигляду для своїх персонажів, які часто є лише косметичними за своєю природою і зазвичай не сприяють прогресу в грі, наприклад просто зміна кольру чи одягу головного героя. Це і є не витратна внутрішньо ігрова покупка. Остаточним типом IAP є **підписки** (**Subscriptions**)- що, мабуть, є більш типовим для звичайних програм на відміну від ігор. Оформлення підписки у основному розблоковує ширший функціонал програми ніж той що доступний при початковоку користуванні. Монетизація за підпискою є хорошим рішенням для меншості програм, таких які послідовно надають постійну цінність для лояльних користувачів за допомогою регулярно оновлюваних функцій або вмісту. Вибір способу монетизація вашої аудиторії є дуже глибокою темою.

Реклама в додатках або IAA(**In App Advertisement**) - це, мабуть, найпопулярніша форма монетизації для інді розробників ігор. Налаштувати її просто, і вона зазвичай не вимагає розвитку складної структури економіки у межах вашої гри. Ви просто інтегруєтесь з наявною рекламною платформою, яка розміщуватиме рекламу у вашій грі - і ви отримувати платежі, якщо ваші користувачі взаємодіють із рекламними оголошеннями. Якщо у вас багато користувачів, це може бути корисною

стратегією. Нагороджене відео, зокрема - є найкращим способом монетизації із усіх інших варіантів що використовуються у IAA. Якщо ви хочете інтегрувати рекламу у свою гру, є багато рекламних мереж. Популярні рекламні рішення включають MoPub, AdColony, Unity Ads, Admob та Adsense. Ми багато говорили про freemium монетизацію, але є багато ігор, які заробляють гроші на мобільному ринку за допомогою старої доброї преміум-моделі. Люди платять один раз вперед за повний досвід гри.

Багато консольних або ПК-ігор, які переносяться на мобільні пристрої, дотримуються цього підходу. Якщо ви перевірите графіки найпопулярніших додатків на iOS та Android ви побачите багато знайомих назв ігор портованих з традиційних ігрових платформ. Платні додатки означає менше часу на розробку подій наживо та у порівнянні з freemium іграми - менше вимог щодо підтримки після запуску.

Схема монетизація обрана мною є змішаною схемою. Є дві версії додатку: платна і бесплатна. У бесплатній версії є реклама та внутрішньо-ігрові не витратні покупки. У платній версії немає реклами, але також є частина внутрішньо-ігрових покупок.

Висновки до другого розділу

У межах другого розділу данох дипломної роботи розглянуто та описано рішення щодо вибору архутектур, рішень та інструментів обраних для розробки додатку. Детально описано операційну систему iOS та пояснено чому саме цю ОС було обрано у якості ОС під яку буде створено додаток. Описано мову програмування Swift , її особливості, плюси, порівняно з її попередниками та обгрунтовано вибір її як мови для розробки додатку. Описано та детально порівняно найпопулярніші підходи та рішення у сфері архітектурних підходів у iOS розробці та обгрунтовано рішення щодо обраного підходу. Описано основні підходи у створенні архітектури користувальницьких інтерфейсів, перечислено особливості побудови інтерфейсів ігрових застосунків. Описано та обгрутовано рішення щодо використання середи розробки та плюси обраної середи. Описано обграні інстременти для створення ресурсів додатку. Описано популярні варіанти монетизації додтаків та обгрунтовано вибір рішення щодо обраних способів монетизації.

РОЗДІЛ 3

ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Інструкція користувача

Додаток склаадється з 15-ти єкранів, кожен з яких виконує свою функцію у додатку. Загальна схема повної архуектури інтерфейсу додатку на зображенні далі (зліва - направо).

Рис 3.1 - Повна архітектура екранів додатку

Перше, що бачить користувач після запуску встановленого додатку - це так знаний «Launch» екран. Цей екран показується користувачеві при кожному запуску додатку поки додаток завантажує ресурси, та виконує код, що має виконуватись при запуску. Мета показу цього єкрану - показ логотипу додатку або копманії розробника. Цей єкран не є інтерактивним та показується приблизно лише пару секунд.

Рис 3.2 - «Launch» екран

У **основі** додатку лежить екран інтерактивної ігрової сцени (перший зліва на схемі повної архтектури). У цей екран завантажується перший або той рівень гри , що користувач зберіг як обраний при останньому виході з додатку. Цей екран відповідає лише за відображення сцени та зчитування натисків користувача на єкран при грі.

Поверх екрану сцени можуть бути показані два типа екранів, а саме: екран головного меню та ігровий екран паузи. Екран головного меню завантажується після старту додатку кожен раз.

Екран головного меню . Головне меню показується поверх екрану ігрової сцени при кожному запуску додатку та після повернення з екрану гри. З головного меню можна потрапити на наступні екрани: **екран налаштувань**, **навчальний екран**, екран **статистики користувача**, **екран магазину**, **екран гри**(паузи).

Рис 3.3 - Екран головного меню

Навчальний екран . Навчальний екран показується поверх головного меню при натисканні користувача на кнопку «?» на екрані головного меню. На на вчальному екрані є пояснення правил гри, та кнопка «не показувати більше» при натиску на котру користувачеві бульше не буде більше показуватись кнопка «?» на екрані головного меню. Більш загальною практикою у розробці мобільних додатків є додавання так званого «Onboarding flow» при першому запуску додатку з повним поясненням функцій та можливостей ,що додоаток надає користувачу. Також загально прийнятим є додавання можливості пропуску цього переліку єкранів, щоб не

змушувати користувачів ,що вже знайомі з додатком переглядати ознайомлюючі екрани знову. У дипломному додатку було вирішено обмежети використання знайомлюйочого сегменту до одного екрану оскільки перелік правил гри, що варто знати користовачу не є великим.

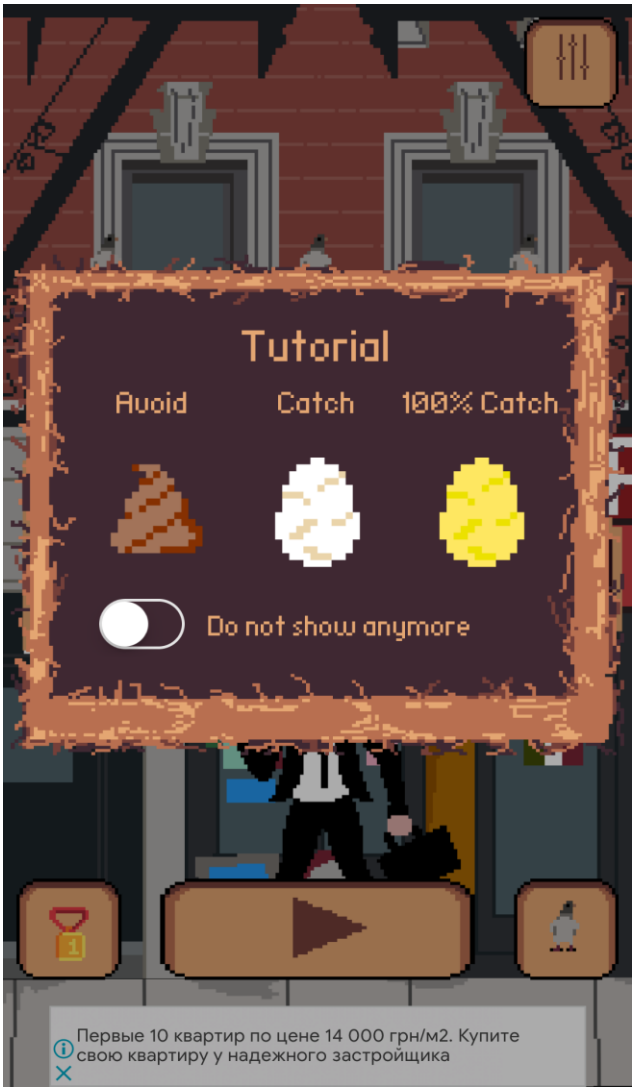


Рис 3.4 - Навчальний екран

Екран налаштувань . Налаштування показуються поверх головного меню при натисканні на кнопку «налаштування». На екрані налаштувань можна змінювати налаштування таких параметрів як: музика, звукові ефекти, мова додатку, руку інтерфеусу. Також можна виконати такі дії як: відновити налаштування проведених внутрішньо-ігрових покупок, оцінити додаток, поділитись додатком та видалити

рекламу з додатку.Також можна перейти на такі екрани як: **екран контактної інформації** та **екран титрів**. Налаштування змінювані користувачем зберігаються локально та відновляться до стандартних при видаленні додатку.

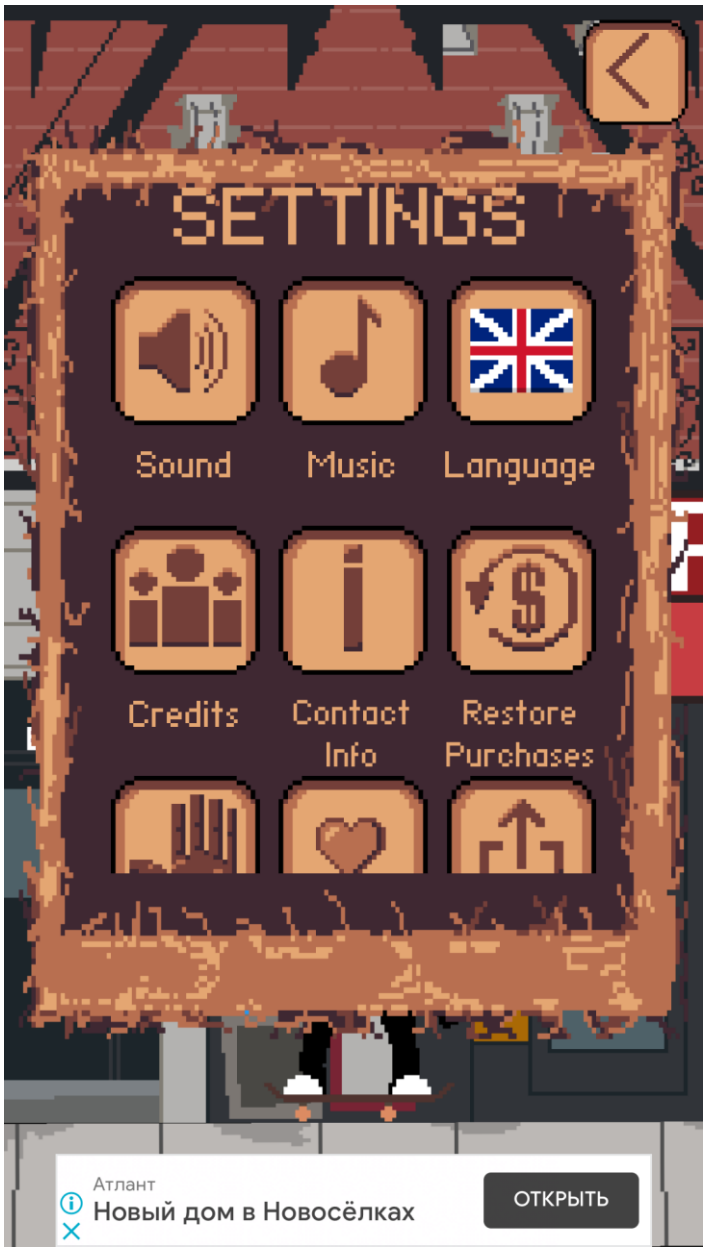


Рис 3.5 - Екран налаштувань

Екран конатктної інформації . Екран контактної інформації показується поверх екрану налаштувань. Якщо користувач девайсу користується стандартним iOS додатком «Пошта» - то можна перейти відразу у готову сторінку електронного листа

для подальшого написання листа та контактування з розробником. Якщо ні - то можна скопіювати контактну інформацію.

Рис 3.6 - Екран контактної інформації

Екран титрів . Екран титрів показується поверх екрану налаштувань. На екрані титрів є інформація джерела яке надало музику для гри, та слова подяки користувачам, що грали. Екран титрів є частково обов’язковим тому що для використання музики за ліцензією треба зазначити контакти представника музики.

Рис 3.7 - Екран титрів

Екран статистики користувача . Екран статистики користувача показується поверх екрану головного меню. На екрані статистики користувача можна переглядати прогрес користувача та завантажувати зображення рівнів у вискокій якості після переглядання відео у бесплатній версії. Якщо користувач без підключення до інтернету спробує відкрити зображення то йому буде покзано екран що треба візобновити підключення.

Рис 3.8 - Екран статистики користувача

Екран магазину . Екран магазину показується поверх екрану головного меню. На екрані магазину можна обрати рівень чи героя та перейти на е**кран** покупки всіх рівнів та героїв . Зі зміною обраного нового рівня чи герою екран ігрової сцени оновлюється та презентує обраний рівень та героя. Зміни можна продивитись не повертаючись на екран гри натиснувши на кнопку «передперегляду».

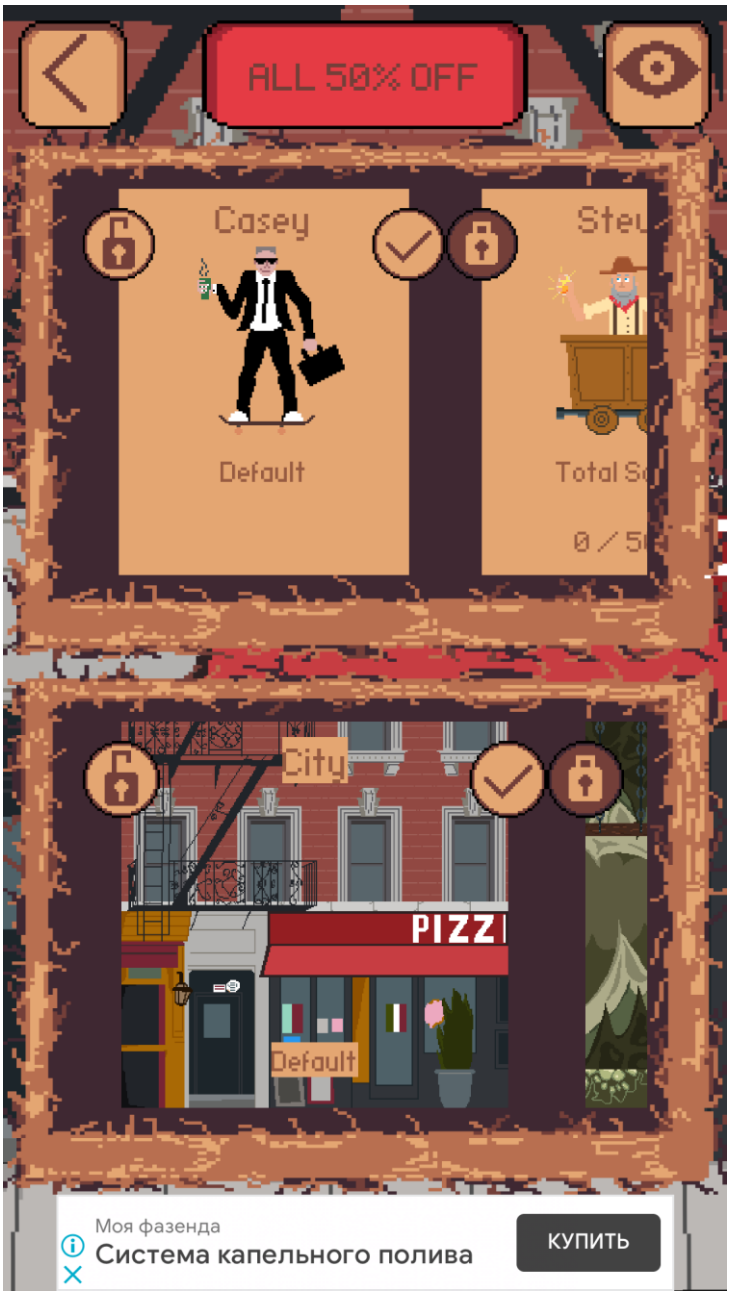


Рис 3.9 - Екран магазину

Екран покупки рівнів та героїв . На екрані покупки рівнів та героїв можна придбати внутрішньо-ігрову покупку.

Рис 3.10- Екран покупки рівнів та героїв

Екран гри . Екран гри показується поверх головного екрану ігрової сцени. На екрані гри показуються результати гри та налаштування паузи, де можна змінювати налаштування музики і звуку.

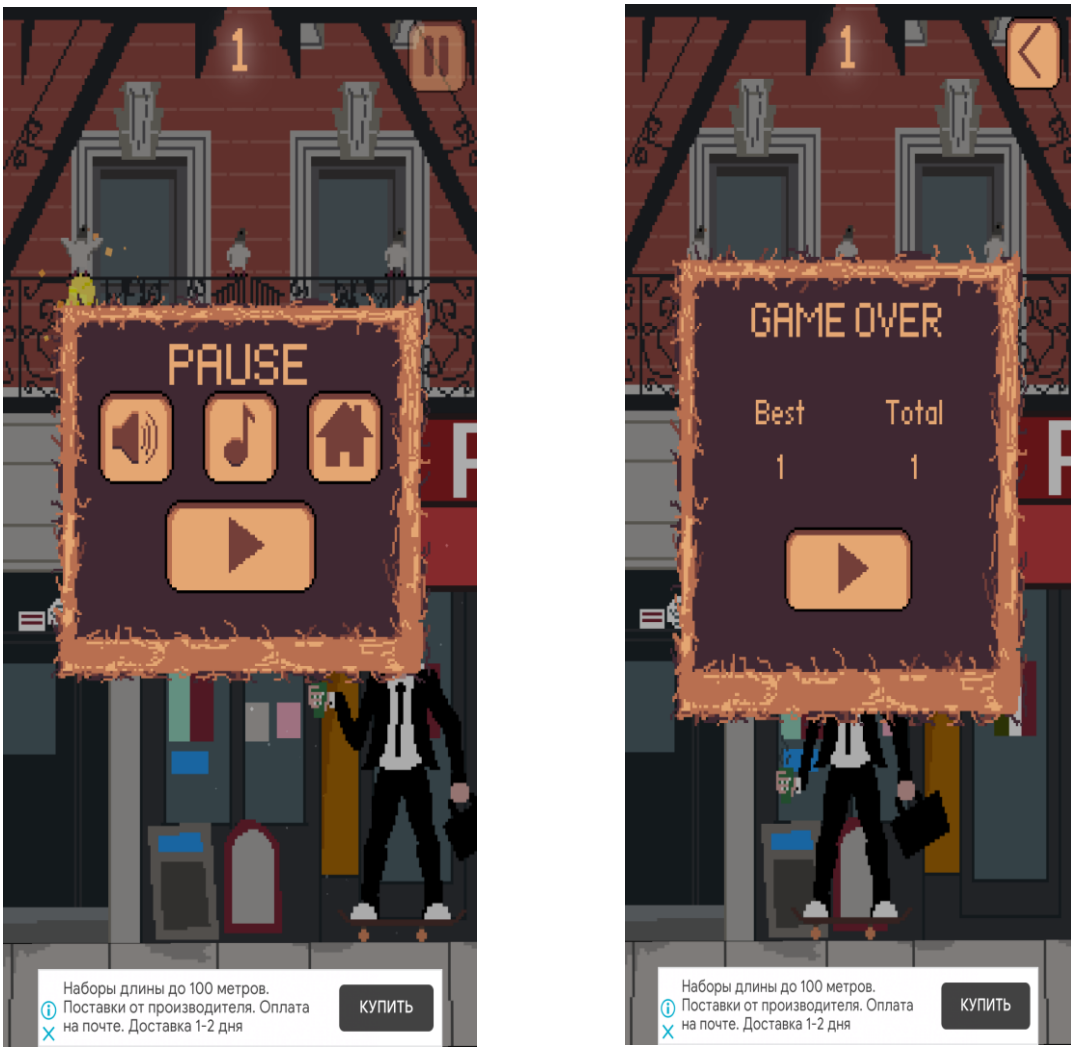


Рис3.11- Екран гри

Висновки до третього розділу

У цьому розділі біло складено детальну інструкцібю користувача для мобільного застосунку під платформу iOS. Кожен крок йде у тому самому порядку, що й коли користувач буде користуватися додатком. Описано структуру екранів та на який з екранів можна потрапити з кожного екрану.

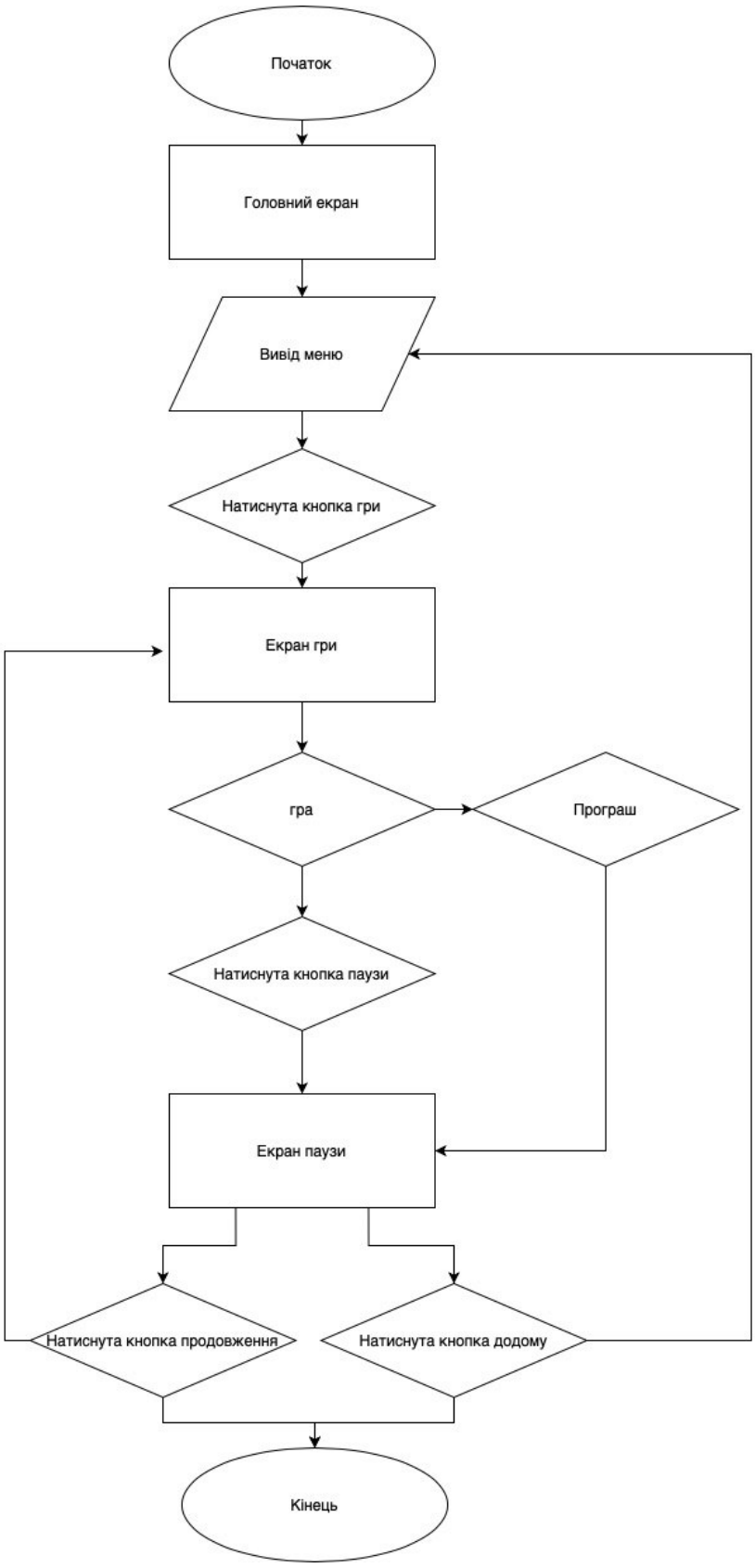
Висновок

Під час виконання дипломного проекту було проаналізовано ринок ігрової індустрії та ринок мобільних ігор. Було проаналізовано найновітніші методології, патерни та інструменти розробки під платформу iOS та не тільки, та обрано список найкращих технологій та інструментів для розробки дипломного додатку.Було розроблено список вимог та задач для розробленого застосукную Було створена поетапна інструкція користувача з поясненням щодо кожного екрану та його можливостей. Розроблений мобільний додаток покрив всі описані вимоги та був опублікований у AppStore.

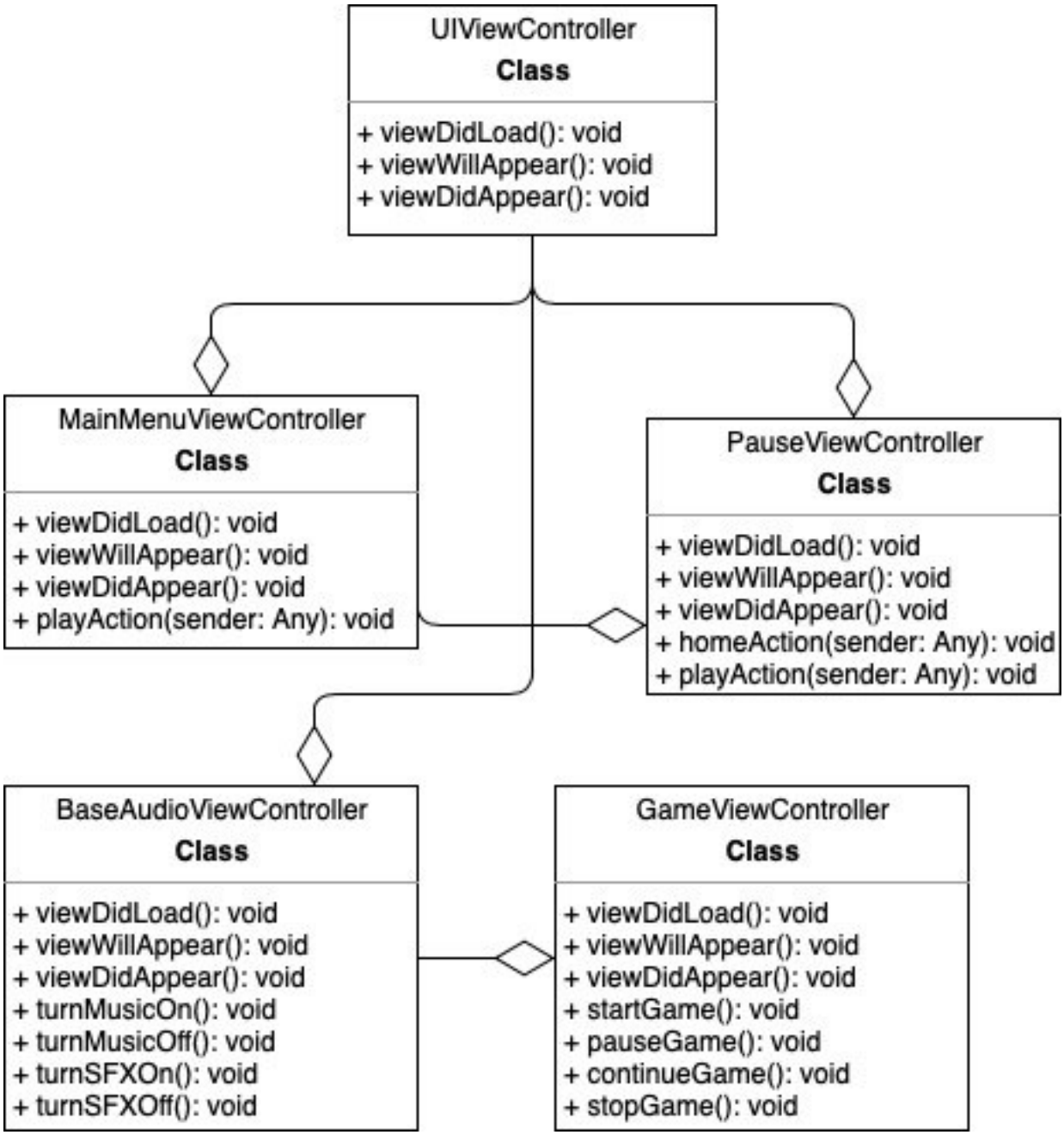
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Pixaki [Електронний ресурс] //Режим доступу: https://rizer.co/pixaki/
2. Procreate [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Procreate_(software)
3. Interface-Builder [Електронний ресурс] //Режим доступу: https://developer.apple.com/xcode/interface-builder/
4. Interface-Builder [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Interface_Builder
5. Xcode [Електронний ресурс] //Режим доступу: https://ru.wikipedia.org/wiki/Xcode
6. Xcode [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Xcode
7. Human Interface Guidelines [Електронний ресурс] //Режим доступу: https://developer.apple.com/design/human-interface-guidelines/ios/app-architecture/navigation/
8. iOS architecture patterns [Електронний ресурс] //Режим доступу: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52#.tliwdfd60
9. Swift programming language [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Swift_(programming_language)
10. IOS [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/IOS
11. Nintendo Switch system software [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Nintendo_Switch_system_software
12. Playstation 4 [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/PlayStation_4
13. Xbox One system software [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Xbox_One_system_software
14. Microsoft Windows [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Microsoft_Windows
15. Android operating system [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Android_(operating_system)
16. Casual game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Casual_game
17. Adventure game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Adventure_game
18. Role-playing game [Електронний ресурс] //Режим доступу: https://en.wikipedia.org/wiki/Role-playing_game

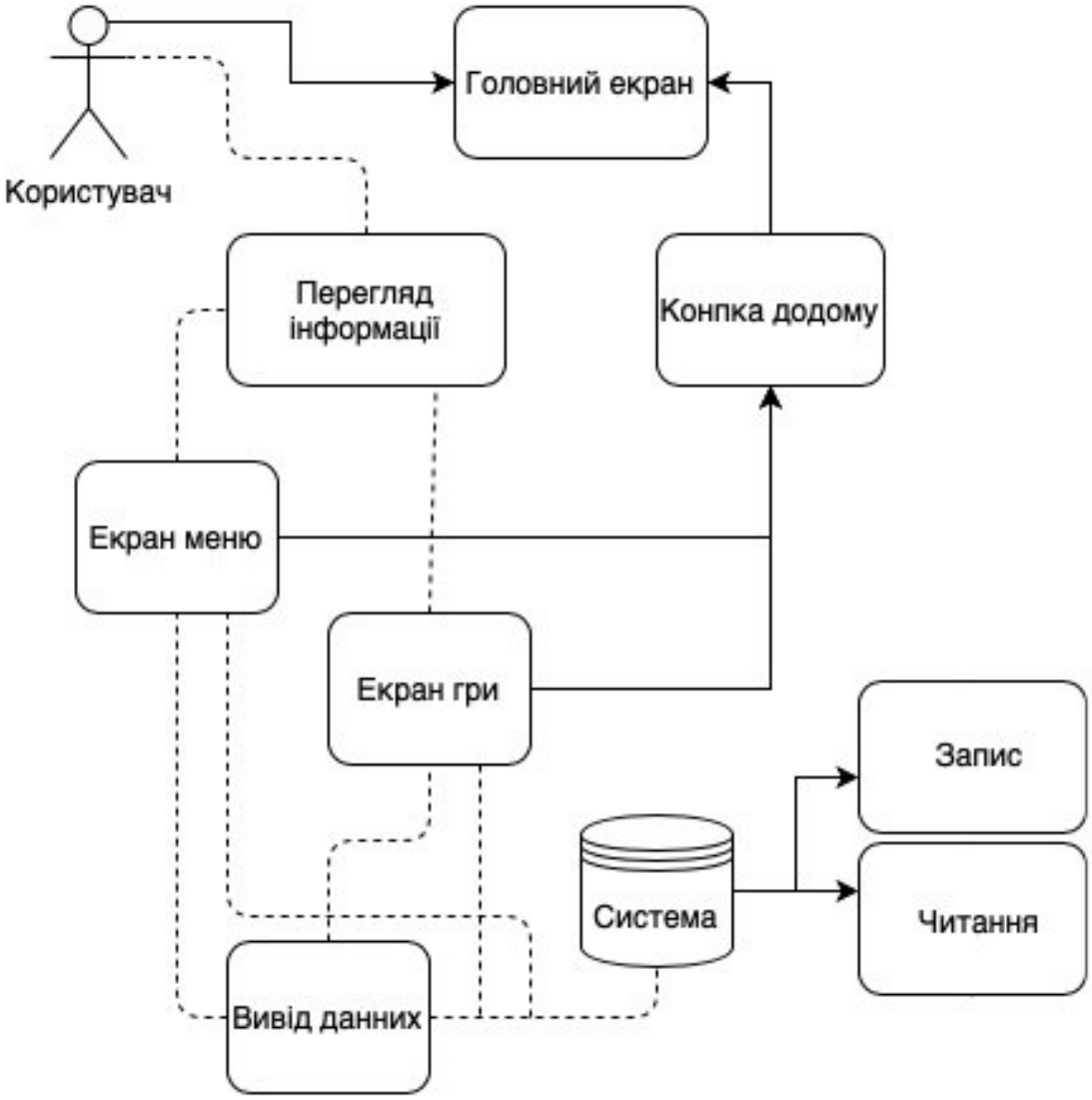
19. Sports game [Електронний ресурс] //Режим доступу:
https://en.wikipedia.org/wiki/Sports_game
20. Strategy game [Електронний ресурс] //Режим доступу:
https://en.wikipedia.org/wiki/Strategy_game



					ІАЛЦ.466500.004 А1		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Поліщук Д.О.*			*Принципова схема алгоритму*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Порєв В.М.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-64*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							



					ІАЛЦ.466500.005 А2		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Поліщук Д.О.*			*Функціональна схема*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Порєв В.М.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-64*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							



					ІАЛЦ.466500.006 А3		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		*Поліщук Д. О.*			*Структурна схема*	Літ.	Аркуш 1 / Аркушів 1
Перевірив		*Порєв В.М.*					
Реценз.						*НТУУ «КПІ», ФІОТ, ІО-64*	
Н. Контр.		*Сімоненко В.П.*					
Затвердив							